BCC342 - INTRODUÇÃO A OTIMIZAÇÃO

Busca Local e Métodos Construtivos

Gladston Juliano Prates Moreira
email: gladston@ufop.edu.br

CSILab, Departamento de Computação
Universidade Federal de Ouro Preto

11 de novembro de 2024

Meta-heurísticas

Heurísticas

Qualquer método aproximado projetado com base nas propriedades estruturais ou nas características das soluções dos problemas, com complexidade reduzida em relação à dos algoritmos exatos e fornecendo, em geral, soluções viáveis de boa qualidade.

- métodos construtivos
- busca local

Meta-heurística

Uma meta-heurística é um procedimento de alto nível ou heurística concebido para encontrar, gerar, ou selecionar um procedimento de nível inferior ou heurística (algoritmo de busca parcial), que pode fornecer uma solução suficientemente boa para um problema de otimização.

UFOP
Universidade Federal de Ouro Preto

Heurísticas

Qualquer método aproximado projetado com base nas propriedades estruturais ou nas características das soluções dos problemas, com complexidade reduzida em relação à dos algoritmos exatos e fornecendo, em geral, soluções viáveis de boa qualidade.

- métodos construtivos
- busca local

Meta-heurística

Uma meta-heurística é um procedimento de alto nível ou heurística concebido para encontrar, gerar, ou selecionar um procedimento de nível inferior ou heurística (algoritmo de busca parcial), que pode fornecer uma solução suficientemente boa para um problema de otimização.

UFOP
Universidade Federal de Ouro Preto

Heurísticas

Qualquer método aproximado projetado com base nas propriedades estruturais ou nas características das soluções dos problemas, com complexidade reduzida em relação à dos algoritmos exatos e fornecendo, em geral, soluções viáveis de boa qualidade.

- métodos construtivos
- busca local

Meta-heurística

Uma meta-heurística é um procedimento de alto nível ou heurística concebido para encontrar, gerar, ou selecionar um procedimento de nível inferior ou heurística (algoritmo de busca parcial), que pode fornecer uma solução suficientemente boa para um problema de otimização.

UFOP
Universidade Federal de Ouro Preto

Heurísticas

Qualquer método aproximado projetado com base nas propriedades estruturais ou nas características das soluções dos problemas, com complexidade reduzida em relação à dos algoritmos exatos e fornecendo, em geral, soluções viáveis de boa qualidade.

- métodos construtivos
- busca local

Meta-heurística

Uma meta-heurística é um procedimento de alto nível ou heurística concebido para encontrar, gerar, ou selecionar um procedimento de nível inferior ou heurística (algoritmo de busca parcial), que pode fornecer uma solução suficientemente boa para um problema de otimização.

Métodos Construtivos

- Consiste em construir uma solução de um problema de forma incremental.
- Passo a passo, um componente é escolhido e depois inserido na solução até gerar uma solução completa.
- O componente escolhido em cada passo é, em geral, o melhor candidato de acordo com algum critério.

UFOP
Universidade Federal de Ouro Preto

Métodos Construtivos

- Consiste em construir uma solução de um problema de forma incremental.
- Passo a passo, um componente é escolhido e depois inserido na solução até gerar uma solução completa.
- O componente escolhido em cada passo é, em geral, o melhor candidato de acordo com algum critério.

Métodos Construtivos

- Consiste em construir uma solução de um problema de forma incremental.
- Passo a passo, um componente é escolhido e depois inserido na solução até gerar uma solução completa.
- O componente escolhido em cada passo é, em geral, o melhor candidato de acordo com algum critério.

Métodos Construtivos

- Consiste em construir uma solução de um problema de forma incremental.
- Passo a passo, um componente é escolhido e depois inserido na solução até gerar uma solução completa.
- O componente escolhido em cada passo é, em geral, o melhor candidato de acordo com algum critério.

Algoritmo Guloso

- Inicialize a solução parcial s_p
- Inicialize o conjunto C de componentes candidatos
- **enquanto** (s_p não for uma solução completa)
  - c = o melhor componente guloso de C
  - sp = s_p + c
  - atualize C
- **fim**
- s = s_p
- retorne s

OBS: S não é necessariamente uma solução ótima do problema.

Algoritmo Guloso

- Inicialize a solução parcial s_p
- Inicialize o conjunto C de componentes candidatos
- **enquanto** (s_p não for uma solução completa)
  - c = o melhor componente guloso de C
  - sp = s_p + c
  - atualize C
- **fim**
- s = s_p
- retorne s

OBS: S não é necessariamente uma solução ótima do problema.

UFOP
Universidade Federal de Ouro Preto

Algoritmo Guloso

- Inicialize a solução parcial s_p
- Inicialize o conjunto C de componentes candidatos
- **enquanto** (s_p não for uma solução completa)
  - c = o melhor componente guloso de C
  - sp = s_p + c
  - atualize C
- **fim**
- s = s_p
- retorne s

OBS: S não é necessariamente uma solução ótima do problema.

Algoritmo Guloso

- Inicialize a solução parcial s_p
- Inicialize o conjunto C de componentes candidatos
- **enquanto** (s_p não for uma solução completa)
  - c = o melhor componente guloso de C
  - sp = s_p + c
  - atualize C
- **fim**
- s = s_p
- retorne s

OBS: S não é necessariamente uma solução ótima do problema.

Algoritmo Guloso

- Inicialize a solução parcial s_p
- Inicialize o conjunto C de componentes candidatos
- **enquanto** (s_p não for uma solução completa)
  - c = o melhor componente guloso de C
  - sp = s_p + c
  - atualize C
- **fim**
- s = s_p
- retorne s

OBS: S não é necessariamente uma solução ótima do problema.

Algoritmo Guloso

- Inicialize a solução parcial s_p
- Inicialize o conjunto C de componentes candidatos
- **enquanto** (s_p não for uma solução completa)
  - c = o melhor componente guloso de C
  - sp = s_p + c
  - atualize C
- **fim**
- s = s_p
- retorne s

OBS: S não é necessariamente uma solução ótima do problema.

Algoritmo Guloso

- Inicialize a solução parcial s_p
- Inicialize o conjunto C de componentes candidatos
- **enquanto** (s_p não for uma solução completa)
  - c = o melhor componente guloso de C
  - sp = s_p + c
  - atualize C
- **fim**
- s = s_p
- retorne s

OBS: S não é necessariamente uma solução ótima do problema.

Algoritmo Guloso

- Inicialize a solução parcial s_p
- Inicialize o conjunto C de componentes candidatos
- **enquanto** (s_p não for uma solução completa)
  - c = o melhor componente guloso de C
  - sp = s_p + c
  - atualize C
- **fim**
- s = s_p
- retorne s

OBS: S não é necessariamente uma solução ótima do problema.

Algoritmo Guloso

- Inicialize a solução parcial s_p
- Inicialize o conjunto C de componentes candidatos
- **enquanto** (s_p não for uma solução completa)
  - c = o melhor componente guloso de C
  - sp = s_p + c
  - atualize C
- **fim**
- s = s_p
- retorne s

OBS: S não é necessariamente uma solução ótima do problema.

UFOP
Universidade Federal de Ouro Preto

Algoritmo Guloso

- Inicialize a solução parcial s_p
- Inicialize o conjunto C de componentes candidatos
- **enquanto** (s_p não for uma solução completa)
  - c = o melhor componente guloso de C
  - sp = s_p + c
  - atualize C
- **fim**
- s = s_p
- retorne s

OBS: S não é necessariamente uma solução ótima do problema.

O Problema da Mochila

- O desafio de encher uma mochila com capacidade limitada, otimizando o valor carregado.
- 1957: Dantzig.
- Relação com inúmeros outros modelos de programação: Investimento de capital; Empacotamento; Carregamento de Veículos; Orçamento, etc.

O problema é formulado como:

$$z = \max \sum_{j=1}^{n} c_j x_j$$

$$sujeito\ a:$$

$$\sum_{j=1}^{n} w_j x_j \leq b$$

$\rightarrow$ as variáveis $x_1, .., x_n$ definem a quantidade de cada objeto que está sendo carregado, cada um com um respectivo peso $w_1, ...w_n$ e valor $c_1, ..., c_n$. A capacidade total da mochila é b.

UFOP
Universidade Federal de Ouro Preto

O Problema da Mochila

- O desafio de encher uma mochila com capacidade limitada, otimizando o valor carregado.
- 1957: Dantzig.
- Relação com inúmeros outros modelos de programação: Investimento de capital; Empacotamento; Carregamento de Veículos; Orçamento, etc.

O problema é formulado como:

$$z = \max \sum_{j=1}^{n} c_j x_j$$

$$\textit{sujeito a}:$$

$$\sum_{j=1}^{n} w_j x_j \leq b$$

$\rightarrow$ as variáveis $x_1, .., x_n$ definem a quantidade de cada objeto que está sendo carregado, cada um com um respectivo peso $w_1, ...w_n$ e valor $c_1, ..., c_n$. A capacidade total da mochila é b.

UFOP
Universidade Federal de Ouro Preto

O Problema da Mochila

- O desafio de encher uma mochila com capacidade limitada, otimizando o valor carregado.
- 1957: Dantzig.
- Relação com inúmeros outros modelos de programação: Investimento de capital; Empacotamento; Carregamento de Veículos; Orçamento, etc.

O problema é formulado como:

$$z = \max \sum_{j=1}^{n} c_j x_j$$

$$sujeito\ a:$$

$$\sum_{j=1}^{n} w_j x_j \leq b$$

$\rightarrow$ as variáveis $x_1, .., x_n$ definem a quantidade de cada objeto que está sendo carregado, cada um com um respectivo peso $w_1, ...w_n$ e valor $c_1, ..., c_n$. A capacidade total da mochila é b.

UFOP
Universidade Federal de Ouro Preto

O Problema da Mochila

- O desafio de encher uma mochila com capacidade limitada, otimizando o valor carregado.
- 1957: Dantzig.
- Relação com inúmeros outros modelos de programação: Investimento de capital; Empacotamento; Carregamento de Veículos; Orçamento, etc.

O problema é formulado como:

$$z = \max \sum_{j=1}^{n} c_j x_j$$
$$\textit{sujeito a} :$$
$$\sum_{j=1}^{n} w_j x_j \leq b$$

$\rightarrow$ as variáveis $x_1, .., x_n$ definem a quantidade de cada objeto que está sendo carregado, cada um com um respectivo peso $w_1, ...w_n$ e valor $c_1, ..., c_n$. A capacidade total da mochila é b.

Algoritmo Guloso - O problema da mochila

Ideia:

Adicionar, em cada passo, o objeto mais valioso por unidade de peso até atingir a capacidade da mochila.

Considere o seguinte problema da mochila:

$$\begin{aligned} \max \quad & f(x) = 3x_1 + 3x_2 + 2x_3 + 4x_4 + 2x_5 + 3x_6 + 5x_7 + 2x_8 \\ \textit{sujeito a}: \quad & \\ & 5x_1 + 4x_2 + 7x_3 + 8x_4 + 4x_5 + 4x_6 + 6x_7 + 8x_8 \leq 25 \\ & \\ & x_j \in \{0, 1\} \end{aligned}$$

UFOP
Universidade Federal de Ouro Preto

Algoritmo Guloso - O problema da mochila

Ideia:

Adicionar, em cada passo, o objeto mais valioso por unidade de peso até atingir a capacidade da mochila.

Considere o seguinte problema da mochila:

$$\begin{aligned} \max \quad & f(x) = 3x_1 + 3x_2 + 2x_3 + 4x_4 + 2x_5 + 3x_6 + 5x_7 + 2x_8 \\ \textit{sujeito a}: \quad & \\ & 5x_1 + 4x_2 + 7x_3 + 8x_4 + 4x_5 + 4x_6 + 6x_7 + 8x_8 \leq 25 \\ & \\ & x_j \in \{0, 1\} \end{aligned}$$

Algoritmo Guloso - O problema da mochila

Ideia:

Adicionar, em cada passo, o objeto mais valioso por unidade de peso até atingir a capacidade da mochila.

Considere o seguinte problema da mochila:

$$\begin{aligned} \max \quad & f(x) = 3x_1 + 3x_2 + 2x_3 + 4x_4 + 2x_5 + 3x_6 + 5x_7 + 2x_8 \\ \textit{sujeito a}: \quad & \\ & 5x_1 + 4x_2 + 7x_3 + 8x_4 + 4x_5 + 4x_6 + 6x_7 + 8x_8 \leq 25 \\ & x_j \in \{0, 1\} \end{aligned}$$

Algoritmo Guloso - O problema da mochila

$objeto(j)$	1	2	3	4	5	6	7	8
$valor(c_j)$	3	3	2	4	2	3	5	2
$peso(w_j)$	5	4	7	8	4	4	6	8

Capacidade da mochila b $= 25$

Função Gulosa (prioridade): $\frac{c_j}{w_j}$

Algoritmo Guloso - O problema da mochila

$objeto(j)$	1	2	3	4	5	6	7	8
$valor(c_j)$	3	3	2	4	2	3	5	2
$peso(w_j)$	5	4	7	8	4	4	6	8

Capacidade da mochila b $= 25$

Função Gulosa (prioridade): $\frac{c_j}{w_j}$

Algoritmo Guloso - O problema da mochila

$objeto(j)$	1	2	3	4	5	6	7	8
$valor(c_j)$	3	3	2	4	2	3	5	2
$peso(w_j)$	5	4	7	8	4	4	6	8
$\frac{c_j}{w_j}$	0,6	0,75	0,29	0,5	0,5	0,75	0,83	0,25

Passo 1

- Quem possui o melhor beneficio por unidade de peso é o **objeto 7**. Logo ele é adicionado na mochila:
- $s_p = (0, 0, 0, 0, 0, 0, 1, 0) \rightarrow$ solução parcial
- peso corrente $= 6 \leq 25$

Algoritmo Guloso - O problema da mochila

$objeto(j)$	1	2	3	4	5	6	7	8
$valor(c_j)$	3	3	2	4	2	3	5	2
$peso(w_j)$	5	4	7	8	4	4	6	8
$\frac{c_j}{w_j}$	0,6	0,75	0,29	0,5	0,5	0,75	0,83	0,25

Passo 1

- Quem possui o melhor beneficio por unidade de peso é o **objeto 7**. Logo ele é adicionado na mochila:
- $s_p = (0, 0, 0, 0, 0, 0, 1, 0) \rightarrow$ solução parcial
- peso corrente $= 6 \leq 25$

Algoritmo Guloso - O problema da mochila

$objeto(j)$	1	2	3	4	5	6	7	8
$valor(c_j)$	3	3	2	4	2	3	5	2
$peso(w_j)$	5	4	7	8	4	4	6	8
$\frac{c_j}{w_j}$	0,6	0,75	0,29	0,5	0,5	0,75	0,83	0,25

Passo 1

- Quem possui o melhor beneficio por unidade de peso é o **objeto 7**. Logo ele é adicionado na mochila:
- $s_p = (0, 0, 0, 0, 0, 0, 1, 0) \rightarrow$ solução parcial
- peso corrente $= 6 \leq 25$

Algoritmo Guloso - O problema da mochila

$objeto(j)$	1	2	3	4	5	6	7	8
$valor(c_j)$	3	3	2	4	2	3	5	2
$peso(w_j)$	5	4	7	8	4	4	6	8
$\frac{c_j}{w_j}$	0,6	0,75	0,29	0,5	0,5	0,75	0,83	0,25

Passo 1

- Quem possui o melhor beneficio por unidade de peso é o **objeto 7**. Logo ele é adicionado na mochila:
- $s_p = (0, 0, 0, 0, 0, 0, 1, 0) \rightarrow$ solução parcial
- peso corrente $= 6 \leq 25$

Algoritmo Guloso - O problema da mochila

$objeto(j)$	1	2	3	4	5	6	7	8
$valor(c_j)$	3	3	2	4	2	3	5	2
$peso(w_j)$	5	4	7	8	4	4	6	8
$\frac{c_j}{w_j}$	0,6	0,75	0,29	0,5	0,5	0,75	0,83	0,25

Passo 2

- A segunda melhor escolha é o **objeto 2**. Logo ele é adicionado na mochila:
- $s_p = (0, 1, 0, 0, 0, 0, 1, 0) \rightarrow$ solução parcial
- peso corrente $= 10 \leq 25$

$objeto(j)$	1	2	3	4	5	6	7	8
$valor(c_j)$	3	3	2	4	2	3	5	2
$peso(w_j)$	5	4	7	8	4	4	6	8
$\frac{c_j}{w_j}$	0,6	0,75	0,29	0,5	0,5	0,75	0,83	0,25

Passo 2

- A segunda melhor escolha é o **objeto 2**. Logo ele é adicionado na mochila:
- $s_p = (0, 1, 0, 0, 0, 0, 1, 0) \rightarrow$ solução parcial
- peso corrente $= 10 \leq 25$

$objeto(j)$	1	2	3	4	5	6	7	8
$valor(c_j)$	3	3	2	4	2	3	5	2
$peso(w_j)$	5	4	7	8	4	4	6	8
$\frac{c_j}{w_j}$	0,6	0,75	0,29	0,5	0,5	0,75	0,83	0,25

Passo 2

- A segunda melhor escolha é o **objeto 2**. Logo ele é adicionado na mochila:
- $s_p = (0, 1, 0, 0, 0, 0, 1, 0) \rightarrow$ solução parcial
- peso corrente $= 10 \leq 25$

Algoritmo Guloso - O problema da mochila

$objeto(j)$	1	2	3	4	5	6	7	8
$valor(c_j)$	3	3	2	4	2	3	5	2
$peso(w_j)$	5	4	7	8	4	4	6	8
$\frac{c_j}{w_j}$	0,6	0,75	0,29	0,5	0,5	0,75	0,83	0,25

Passo 2

- A segunda melhor escolha é o **objeto 2**. Logo ele é adicionado na mochila:
- $s_p = (0, 1, 0, 0, 0, 0, 1, 0) \rightarrow$ solução parcial
- peso corrente $= 10 \leq 25$

Algoritmo Guloso - O problema da mochila

$objeto(j)$	1	2	3	4	5	6	7	8
$valor(c_j)$	3	3	2	4	2	3	5	2
$peso(w_j)$	5	4	7	8	4	4	6	8
$\frac{c_j}{w_j}$	0,6	0,75	0,29	0,5	0,5	0,75	0,83	0,25

Passo 3

- A terceira melhor escolha é o **objeto 6**. Logo ele é adicionado na mochila:
- $s_p = (0, 1, 0, 0, 0, 1, 1, 0) \rightarrow$ solução parcial
- peso corrente $= 14 \leq 25$

$objeto(j)$	1	2	3	4	5	6	7	8
$valor(c_j)$	3	3	2	4	2	3	5	2
$peso(w_j)$	5	4	7	8	4	4	6	8
$\frac{c_j}{w_j}$	0,6	0,75	0,29	0,5	0,5	0,75	0,83	0,25

Passo 3

- A terceira melhor escolha é o **objeto 6**. Logo ele é adicionado na mochila:
- $s_p = (0, 1, 0, 0, 0, 1, 1, 0) \rightarrow$ solução parcial
- peso corrente $= 14 \leq 25$

Algoritmo Guloso - O problema da mochila

$objeto(j)$	1	2	3	4	5	6	7	8
$valor(c_j)$	3	3	2	4	2	3	5	2
$peso(w_j)$	5	4	7	8	4	4	6	8
$\frac{c_j}{w_j}$	0,6	0,75	0,29	0,5	0,5	0,75	0,83	0,25

Passo 3

- A terceira melhor escolha é o **objeto 6**. Logo ele é adicionado na mochila:
- $s_p = (0, 1, 0, 0, 0, 1, 1, 0) \rightarrow$ solução parcial
- peso corrente $= 14 \leq 25$

$objeto(j)$	1	2	3	4	5	6	7	8
$valor(c_j)$	3	3	2	4	2	3	5	2
$peso(w_j)$	5	4	7	8	4	4	6	8
$\frac{c_j}{w_j}$	0,6	0,75	0,29	0,5	0,5	0,75	0,83	0,25

Passo 3

- A terceira melhor escolha é o **objeto 6**. Logo ele é adicionado na mochila:
- $s_p = (0, 1, 0, 0, 0, 1, 1, 0) \rightarrow$ solução parcial
- peso corrente $= 14 \leq 25$

Algoritmo Guloso - O problema da mochila

$objeto(j)$	1	2	3	4	5	6	7	8
$valor(c_j)$	3	3	2	4	2	3	5	2
$peso(w_j)$	5	4	7	8	4	4	6	8
$\frac{c_j}{w_j}$	0,6	0,75	0,29	0,5	0,5	0,75	0,83	0,25

Passo 4

- A quarta melhor escolha é o **objeto 1**. Logo ele é adicionado na mochila:
- $s_p = (1, 1, 0, 0, 0, 1, 1, 0) \rightarrow$ solução parcial
- peso corrente $= 19 \leq 25$

Algoritmo Guloso - O problema da mochila

$objeto(j)$	1	2	3	4	5	6	7	8
$valor(c_j)$	3	3	2	4	2	3	5	2
$peso(w_j)$	5	4	7	8	4	4	6	8
$\frac{c_j}{w_j}$	0,6	0,75	0,29	0,5	0,5	0,75	0,83	0,25

Passo 4

- A quarta melhor escolha é o **objeto 1**. Logo ele é adicionado na mochila:
- $s_p = (1, 1, 0, 0, 0, 1, 1, 0) \rightarrow$ solução parcial
- peso corrente $= 19 \leq 25$

Algoritmo Guloso - O problema da mochila

$objeto(j)$	1	2	3	4	5	6	7	8
$valor(c_j)$	3	3	2	4	2	3	5	2
$peso(w_j)$	5	4	7	8	4	4	6	8
$\frac{c_j}{w_j}$	0,6	0,75	0,29	0,5	0,5	0,75	0,83	0,25

Passo 4

- A quarta melhor escolha é o **objeto 1**. Logo ele é adicionado na mochila:
- $s_p = (1, 1, 0, 0, 0, 1, 1, 0) \rightarrow$ solução parcial
- peso corrente $= 19 \leq 25$

$objeto(j)$	1	2	3	4	5	6	7	8
$valor(c_j)$	3	3	2	4	2	3	5	2
$peso(w_j)$	5	4	7	8	4	4	6	8
$\frac{c_j}{w_j}$	0,6	0,75	0,29	0,5	0,5	0,75	0,83	0,25

Passo 4

- A quarta melhor escolha é o **objeto 1**. Logo ele é adicionado na mochila:
- $s_p = (1, 1, 0, 0, 0, 1, 1, 0) \rightarrow$ solução parcial
- peso corrente $= 19 \leq 25$

Algoritmo Guloso - O problema da mochila

$objeto(j)$	1	2	3	4	5	6	7	8
$valor(c_j)$	3	3	2	4	2	3	5	2
$peso(w_j)$	5	4	7	8	4	4	6	8
$\frac{c_j}{w_j}$	0,6	0,75	0,29	0,5	0,5	0,75	0,83	0,25

Passo 5

- Como não é possível adicionar o objeto 4, adicionamos o **objeto 5**. Logo ele é adicionado na mochila:
- $s_p = (1, 1, 0, 0, 1, 1, 1, 0) \rightarrow$ solução parcial
- peso corrente $= 23 \leq 25$
- Não é possível mais adicionar objetos!
- Solução final $s = (1, 1, 0, 0, 1, 1, 1, 0) \rightarrow$ valor $= 16$.

Algoritmo Guloso - O problema da mochila

$objeto(j)$	1	2	3	4	5	6	7	8
$valor(c_j)$	3	3	2	4	2	3	5	2
$peso(w_j)$	5	4	7	8	4	4	6	8
$\frac{c_j}{w_j}$	0,6	0,75	0,29	0,5	0,5	0,75	0,83	0,25

Passo 5

- Como não é possível adicionar o objeto 4, adicionamos o **objeto 5**. Logo ele é adicionado na mochila:
- $s_p = (1, 1, 0, 0, 1, 1, 1, 0) \rightarrow$ solução parcial
- peso corrente $= 23 \leq 25$
- Não é possível mais adicionar objetos!
- Solução final $s = (1, 1, 0, 0, 1, 1, 1, 0) \rightarrow$ valor $= 16$.

Algoritmo Guloso - O problema da mochila

$objeto(j)$	1	2	3	4	5	6	7	8
$valor(c_j)$	3	3	2	4	2	3	5	2
$peso(w_j)$	5	4	7	8	4	4	6	8
$\frac{c_j}{w_j}$	0,6	0,75	0,29	0,5	0,5	0,75	0,83	0,25

Passo 5

- Como não é possível adicionar o objeto 4, adicionamos o **objeto 5**. Logo ele é adicionado na mochila:
- $s_p = (1, 1, 0, 0, 1, 1, 1, 0) \rightarrow$ solução parcial
- peso corrente $= 23 \leq 25$
- Não é possível mais adicionar objetos!
- Solução final $s = (1, 1, 0, 0, 1, 1, 1, 0) \rightarrow$ valor $= 16$.

Algoritmo Guloso - O problema da mochila

$objeto(j)$	1	2	3	4	5	6	7	8
$valor(c_j)$	3	3	2	4	2	3	5	2
$peso(w_j)$	5	4	7	8	4	4	6	8
$\frac{c_j}{w_j}$	0,6	0,75	0,29	0,5	0,5	0,75	0,83	0,25

Passo 5

- Como não é possível adicionar o objeto 4, adicionamos o **objeto 5**. Logo ele é adicionado na mochila:
- $s_p = (1, 1, 0, 0, 1, 1, 1, 0) \rightarrow$ solução parcial
- peso corrente $= 23 \leq 25$
- Não é possível mais adicionar objetos!
- Solução final $s = (1, 1, 0, 0, 1, 1, 1, 0) \rightarrow$ valor $= 16$.

Algoritmo Guloso - O problema da mochila

$objeto(j)$	1	2	3	4	5	6	7	8
$valor(c_j)$	3	3	2	4	2	3	5	2
$peso(w_j)$	5	4	7	8	4	4	6	8
$\frac{c_j}{w_j}$	0,6	0,75	0,29	0,5	0,5	0,75	0,83	0,25

Passo 5

- Como não é possível adicionar o objeto 4, adicionamos o **objeto 5**. Logo ele é adicionado na mochila:
- $s_p = (1, 1, 0, 0, 1, 1, 1, 0) \rightarrow$ solução parcial
- peso corrente $= 23 \leq 25$
- Não é possível mais adicionar objetos!
- Solução final $s = (1, 1, 0, 0, 1, 1, 1, 0) \rightarrow$ valor $= 16$.

Algoritmo Guloso - O problema da mochila

$objeto(j)$	1	2	3	4	5	6	7	8
$valor(c_j)$	3	3	2	4	2	3	5	2
$peso(w_j)$	5	4	7	8	4	4	6	8
$\frac{c_j}{w_j}$	0,6	0,75	0,29	0,5	0,5	0,75	0,83	0,25

Passo 5

- Como não é possível adicionar o objeto 4, adicionamos o **objeto 5**. Logo ele é adicionado na mochila:
- $s_p = (1, 1, 0, 0, 1, 1, 1, 0) \rightarrow$ solução parcial
- peso corrente $= 23 \leq 25$
- Não é possível mais adicionar objetos!
- Solução final $s = (1, 1, 0, 0, 1, 1, 1, 0) \rightarrow$ valor $= 16$.

UFOP
Universidade Federal de Ouro Preto

O problema do caixeiro viajante

Definição

Dada uma lista de cidades e as distâncias entre cada par de cidades, qual é a rota mais curta que visita cada cidade exatamente uma vez e retorna para a cidade de origem?

Formulção

Sejam

- $n =$ número de cidades
- $c_{ij} =$ o custo de se viajar da cidade i para a cidade j
- $x_{ij} = \begin{cases} 1, \text{se existe o caminho de } i \text{para } j \\ 0, \text{caso contrário} \end{cases}$

O problema do caixeiro viajante

Definição

Dada uma lista de cidades e as distâncias entre cada par de cidades, qual é a rota mais curta que visita cada cidade exatamente uma vez e retorna para a cidade de origem?

Formulção

Sejam

- $n =$ número de cidades
- $c_{ij} =$ o custo de se viajar da cidade i para a cidade j
- $x_{ij} = \begin{cases} 1, \text{se existe o caminho de } i \text{para } j \\ 0, \text{caso contrário} \end{cases}$

O problema do caixeiro viajante

Definição

Dada uma lista de cidades e as distâncias entre cada par de cidades, qual é a rota mais curta que visita cada cidade exatamente uma vez e retorna para a cidade de origem?

Formulção

Sejam

- $n =$ número de cidades
- $c_{ij} =$ o custo de se viajar da cidade i para a cidade j
- $x_{ij} = \begin{cases} 1, \text{se existe o caminho de } i \text{para } j \\ 0, \text{caso contrário} \end{cases}$

O problema do caixeiro viajante

Definição

Dada uma lista de cidades e as distâncias entre cada par de cidades, qual é a rota mais curta que visita cada cidade exatamente uma vez e retorna para a cidade de origem?

Formulção

Sejam

- $n =$ número de cidades
- $c_{ij} =$ o custo de se viajar da cidade i para a cidade j
- $x_{ij} = \begin{cases} 1, \text{se existe o caminho de } i \text{para } j \\ 0, \text{caso contrário} \end{cases}$

Modelo - PCV

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

$$\begin{aligned} \textit{sujeito a}: \quad & \textstyle\sum_{i=1}^{n} x_{ij} = 1 \quad j = 1, \ldots, n \\ & \textstyle\sum_{j=1}^{n} x_{ij} = 1 \quad i = 1, \ldots, n \\ & x_{ij} \in \{0, 1\} \qquad i, j = 1, \ldots, n \end{aligned}$$

- Para N cidades existem $(N-1)!$ rotas possíveis.
- Se $N = 11$, existem $10! = 3.628.800$ rotas possíveis.
- Se $N = 26$, existem $25! = 15.511.210.043.330.985.984.000.000$ rotas possíveis.

Modelo - PCV

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

$$\begin{aligned} \textit{sujeito a}: \quad & \textstyle\sum_{i=1}^{n} x_{ij} = 1 \quad j = 1, \ldots, n \\ & \textstyle\sum_{j=1}^{n} x_{ij} = 1 \quad i = 1, \ldots, n \\ & x_{ij} \in \{0, 1\} \qquad i, j = 1, \ldots, n \end{aligned}$$

- Para N cidades existem $(N-1)!$ rotas possíveis.
- Se $N = 11$, existem $10! = 3.628.800$ rotas possíveis.
- Se $N = 26$, existem $25! = 15.511.210.043.330.985.984.000.000$ rotas possíveis.

Modelo - PCV

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

$$\begin{aligned} \textit{sujeito a}: \quad & \textstyle\sum_{i=1}^{n} x_{ij} = 1 \quad j = 1, \ldots, n \\ & \textstyle\sum_{j=1}^{n} x_{ij} = 1 \quad i = 1, \ldots, n \\ & x_{ij} \in \{0, 1\} \qquad i, j = 1, \ldots, n \end{aligned}$$

- Para N cidades existem $(N-1)!$ rotas possíveis.
- Se $N = 11$, existem $10! = 3.628.800$ rotas possíveis.
- Se $N = 26$, existem $25! = 15.511.210.043.330.985.984.000.000$ rotas possíveis.

Modelo - PCV

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

$$\begin{aligned} \textit{sujeito a}: \quad & \textstyle\sum_{i=1}^{n} x_{ij} = 1 \quad j = 1, \ldots, n \\ & \textstyle\sum_{j=1}^{n} x_{ij} = 1 \quad i = 1, \ldots, n \\ & x_{ij} \in \{0, 1\} \qquad i, j = 1, \ldots, n \end{aligned}$$

- Para N cidades existem $(N-1)!$ rotas possíveis.
- Se $N = 11$, existem $10! = 3.628.800$ rotas possíveis.
- Se $N = 26$, existem $25! = 15.511.210.043.330.985.984.000.000$ rotas possíveis.

O problema do Caixeiro Viajante

Algoritmo Guloso - PCV - vizinhos mais próximos

Escolha uma cidade inicial aleatoriamente. Adicione, passo a passo, uma nova cidade. Em cada passo, escolha, entre as cidades que restaram, a cidade mais próxima.

- Influência do nó inicial.
- Possibilidade de não encontrar uma solução viável
- Possibilidade de obter soluções arbitrariamente ruins

UFOP
Universidade Federal de Ouro Preto

O problema do Caixeiro Viajante

Algoritmo Guloso - PCV - vizinhos mais próximos

Escolha uma cidade inicial aleatoriamente. Adicione, passo a passo, uma nova cidade. Em cada passo, escolha, entre as cidades que restaram, a cidade mais próxima.

- Influência do nó inicial.
- Possibilidade de não encontrar uma solução viável
- Possibilidade de obter soluções arbitrariamente ruins

O problema do Caixeiro Viajante

Algoritmo Guloso - PCV - vizinhos mais próximos

Escolha uma cidade inicial aleatoriamente. Adicione, passo a passo, uma nova cidade. Em cada passo, escolha, entre as cidades que restaram, a cidade mais próxima.

- Influência do nó inicial.
- Possibilidade de não encontrar uma solução viável
- Possibilidade de obter soluções arbitrariamente ruins

UFOP
Universidade Federal de Ouro Preto

O problema do Caixeiro Viajante

Algoritmo Guloso - PCV - vizinhos mais próximos

Escolha uma cidade inicial aleatoriamente. Adicione, passo a passo, uma nova cidade. Em cada passo, escolha, entre as cidades que restaram, a cidade mais próxima.

- Influência do nó inicial.
- Possibilidade de não encontrar uma solução viável
- Possibilidade de obter soluções arbitrariamente ruins

UFOP
Universidade Federal de Ouro Preto

Algoritmo Guloso - PCV - vizinhos mais próximos

Considere o PCV

Com matriz de distâncias entre 5 cidades:

	1	**2**	**3**	**4**	**5**
1	0	1	2,2	2	4,1
2	1	0	1,4	2,2	4
3	2,2	1,4	0	2,2	3,2
4	2	2,2	2,2	0	2,2
5	4,1	4	3,2	2,2	0

- Neste caso, existem $(4)! = 24$ rotas possíveis.
- Escolhe a **cidade 1**, para construir a solução gulosa.

Algoritmo Guloso - PCV - vizinhos mais próximos

Considere o PCV

Com matriz de distâncias entre 5 cidades:

	1	**2**	**3**	**4**	**5**
1	0	1	2,2	2	4,1
2	1	0	1,4	2,2	4
3	2,2	1,4	0	2,2	3,2
4	2	2,2	2,2	0	2,2
5	4,1	4	3,2	2,2	0

- Neste caso, existem $(4)! = 24$ rotas possíveis.
- Escolhe a **cidade 1**, para construir a solução gulosa.

Algoritmo Guloso - PCV - vizinhos mais próximos

Considere o PCV

Com matriz de distâncias entre 5 cidades:

	1	**2**	**3**	**4**	**5**
1	0	1	2,2	2	4,1
2	1	0	1,4	2,2	4
3	2,2	1,4	0	2,2	3,2
4	2	2,2	2,2	0	2,2
5	4,1	4	3,2	2,2	0

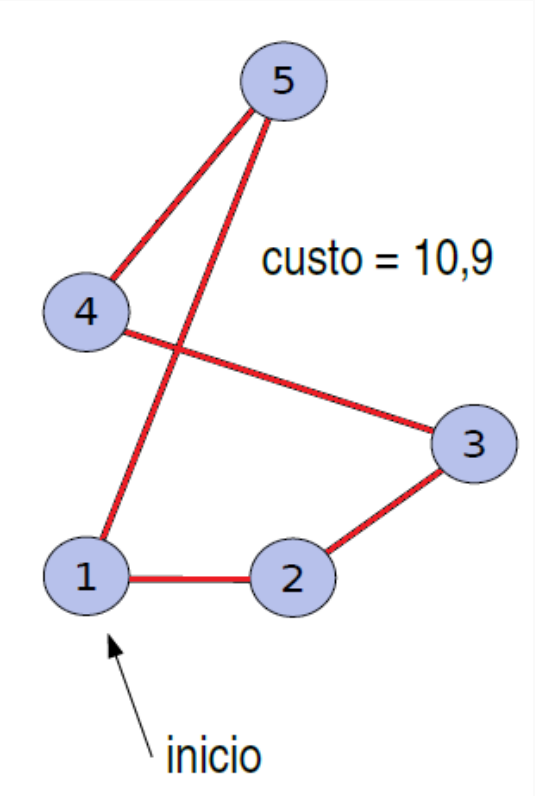
- Neste caso, existem $(4)! = 24$ rotas possíveis.
- Escolhe a **cidade 1**, para construir a solução gulosa.

Algoritmo Guloso - PCV - vizinhos mais próximos

Considere o PCV

Com matriz de distâncias entre 5 cidades:

	1	**2**	**3**	**4**	**5**
1	0	1	2,2	2	4,1
2	1	0	1,4	2,2	4
3	2,2	1,4	0	2,2	3,2
4	2	2,2	2,2	0	2,2
5	4,1	4	3,2	2,2	0

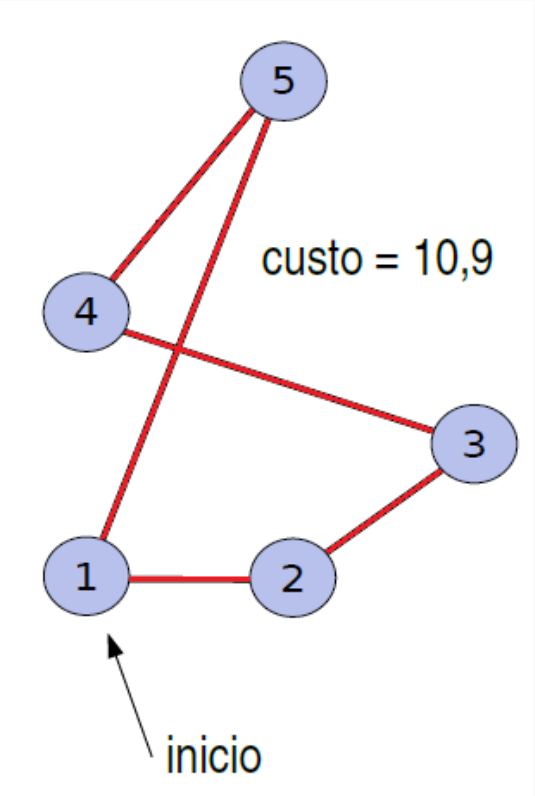


Algoritmo Guloso - PCV - vizinhos mais próximos

Considere o PCV

Com matriz de distâncias entre 5 cidades:

	1	**2**	**3**	**4**	**5**
1	0	1	2,2	2	4,1
2	1	0	1,4	2,2	4
3	2,2	1,4	0	2,2	3,2
4	2	2,2	2,2	0	2,2
5	4,1	4	3,2	2,2	0



O problema do Caixeiro Viajante

Algoritmo Guloso - PCV - inserção mais barata

- Iniciar com um (ciclo).
- Inserir uma cidade k entre as cidades i e j tal que minimize $s_{ij} = c_{ik} + c_{kj} - c_{ij}$.
- Inserir as arestas (i, k) e (k, j) e retirar a aresta (i, j).
- Repita o passo 2 até que todas as cidades sejam visitadas.

O problema do Caixeiro Viajante

Algoritmo Guloso - PCV - inserção mais barata

- Iniciar com um (ciclo).
- Inserir uma cidade k entre as cidades i e j tal que minimize $s_{ij} = c_{ik} + c_{kj} - c_{ij}$.
- Inserir as arestas (i, k) e (k, j) e retirar a aresta (i, j).
- Repita o passo 2 até que todas as cidades sejam visitadas.

O problema do Caixeiro Viajante

Algoritmo Guloso - PCV - inserção mais barata

- Iniciar com um (ciclo).
- Inserir uma cidade k entre as cidades i e j tal que minimize $s_{ij} = c_{ik} + c_{kj} - c_{ij}$.
- Inserir as arestas (i, k) e (k, j) e retirar a aresta (i, j).
- Repita o passo 2 até que todas as cidades sejam visitadas.

O problema do Caixeiro Viajante

Algoritmo Guloso - PCV - inserção mais barata

- Iniciar com um (ciclo).
- Inserir uma cidade k entre as cidades i e j tal que minimize $s_{ij} = c_{ik} + c_{kj} - c_{ij}$.
- Inserir as arestas (i, k) e (k, j) e retirar a aresta (i, j).
- Repita o passo 2 até que todas as cidades sejam visitadas.

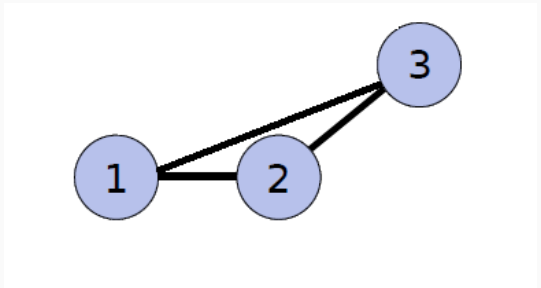
Algoritmo Guloso - PCV - inserção mais barata

Considere o PCV

Com matriz de distâncias entre 5 cidades:

	1	**2**	**3**	**4**	**5**
1	0	1	2,2	2	4,1
2	1	0	1,4	2,2	4
3	2,2	1,4	0	2,2	3,2
4	2	2,2	2,2	0	2,2
5	4,1	4	3,2	2,2	0

- Ciclo inicial: pode ser escolhido, por exemplo, pela heurística dos vizinhos mais próximos.



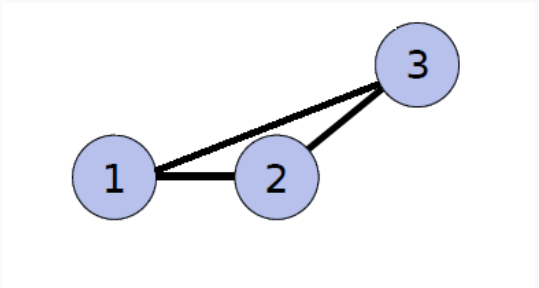
Algoritmo Guloso - PCV - inserção mais barata

Considere o PCV

Com matriz de distâncias entre 5 cidades:

	1	**2**	**3**	**4**	**5**
1	0	1	2,2	2	4,1
2	1	0	1,4	2,2	4
3	2,2	1,4	0	2,2	3,2
4	2	2,2	2,2	0	2,2
5	4,1	4	3,2	2,2	0

- Ciclo inicial: pode ser escolhido, por exemplo, pela heurística dos vizinhos mais próximos.



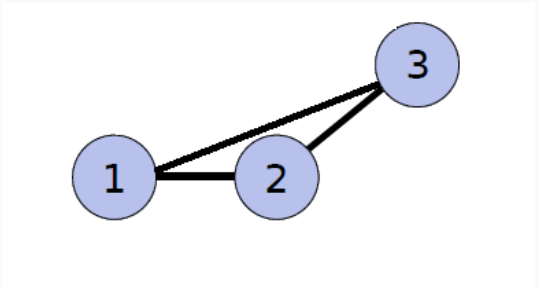
Algoritmo Guloso - PCV - inserção mais barata

Considere o PCV

Com matriz de distâncias entre 5 cidades:

	1	**2**	**3**	**4**	**5**
1	0	1	2,2	2	4,1
2	1	0	1,4	2,2	4
3	2,2	1,4	0	2,2	3,2
4	2	2,2	2,2	0	2,2
5	4,1	4	3,2	2,2	0

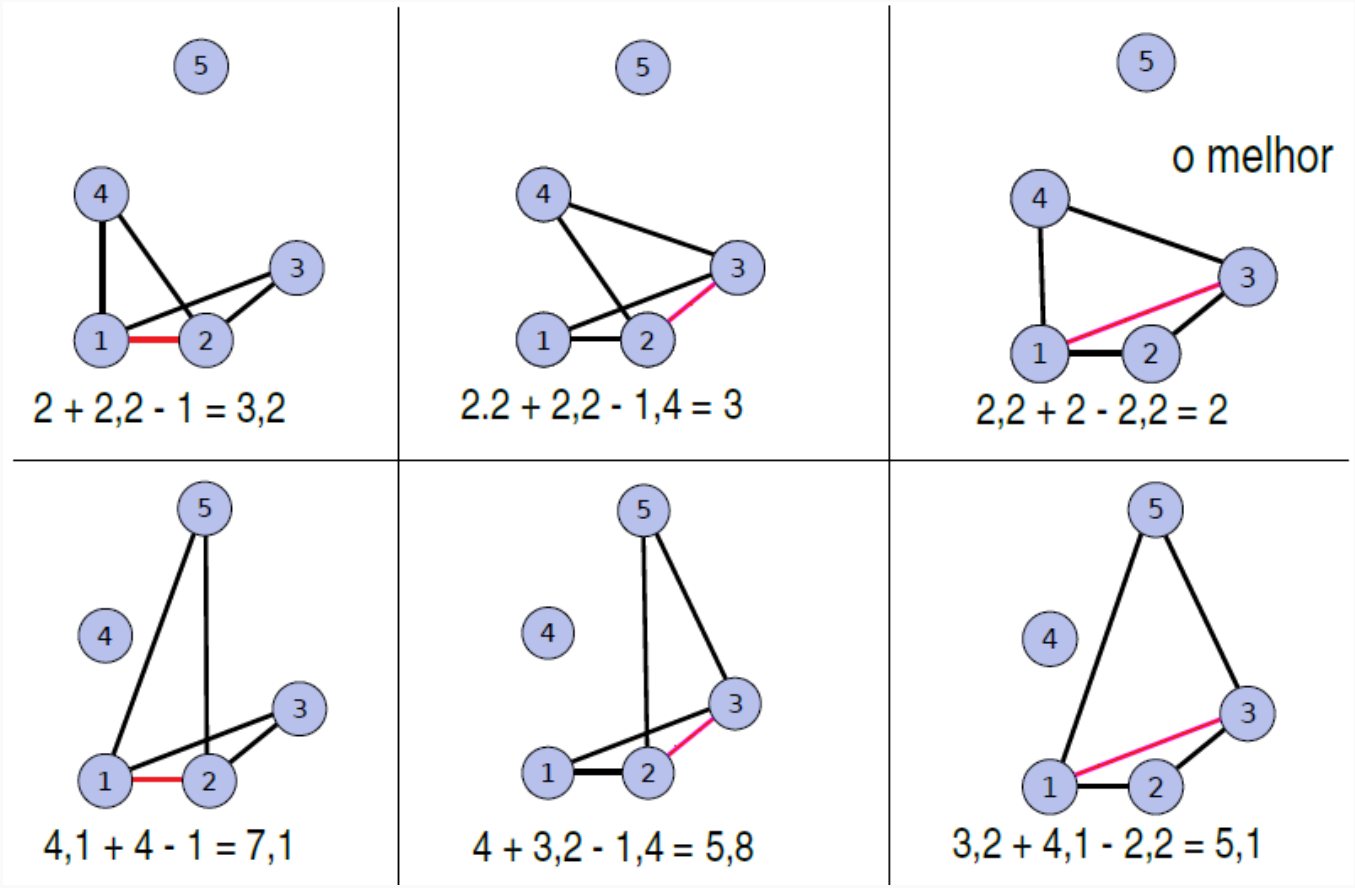
- Ciclo inicial: pode ser escolhido, por exemplo, pela heurística dos vizinhos mais próximos.



Algoritmo Guloso - PCV - inserção mais barata

Encontrar k entre as cidades i e j tal que minimize

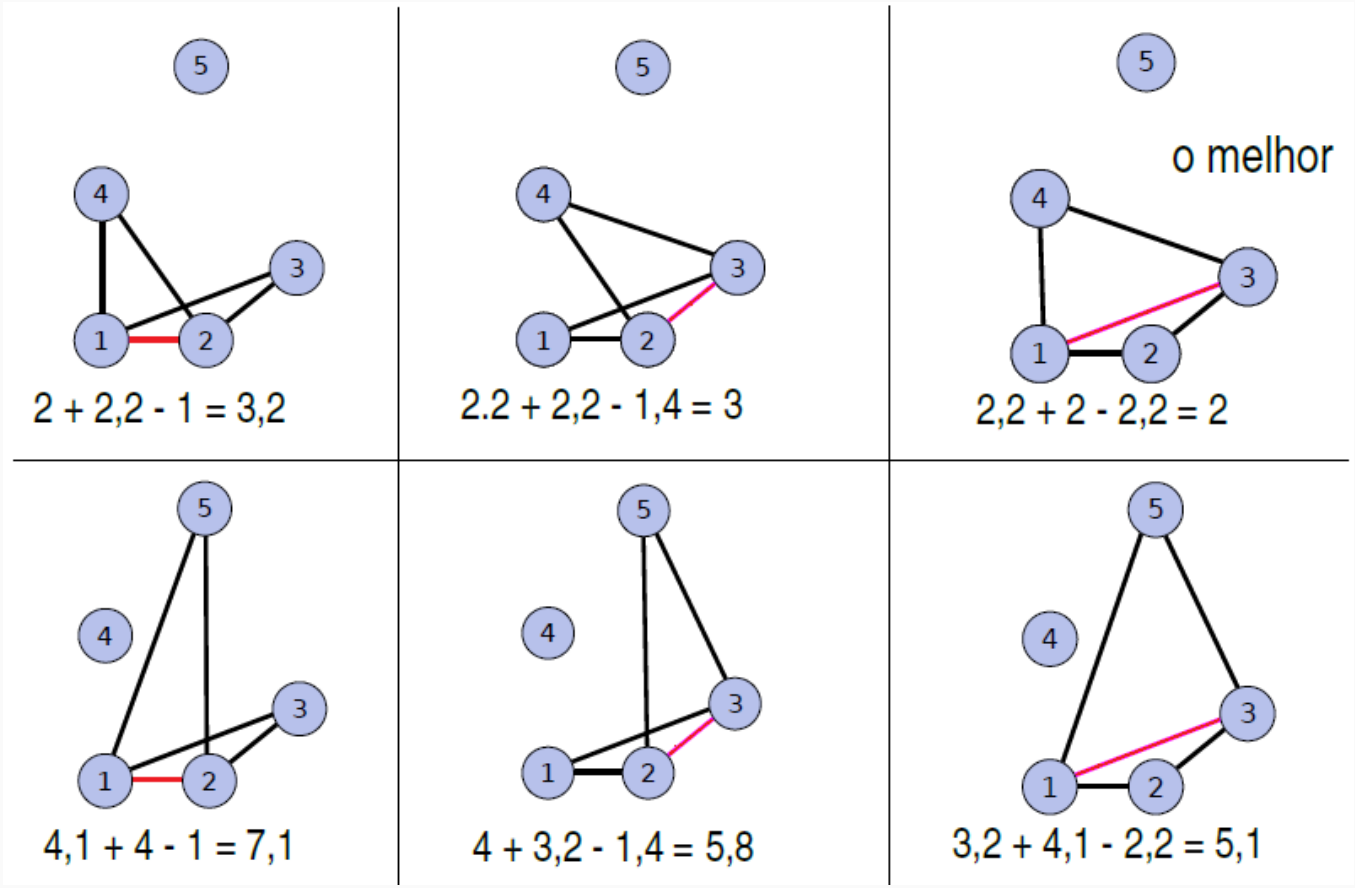
$s_{ij} = c_{ik} + c_{kj} - c_{ij}$



Algoritmo Guloso - PCV - inserção mais barata

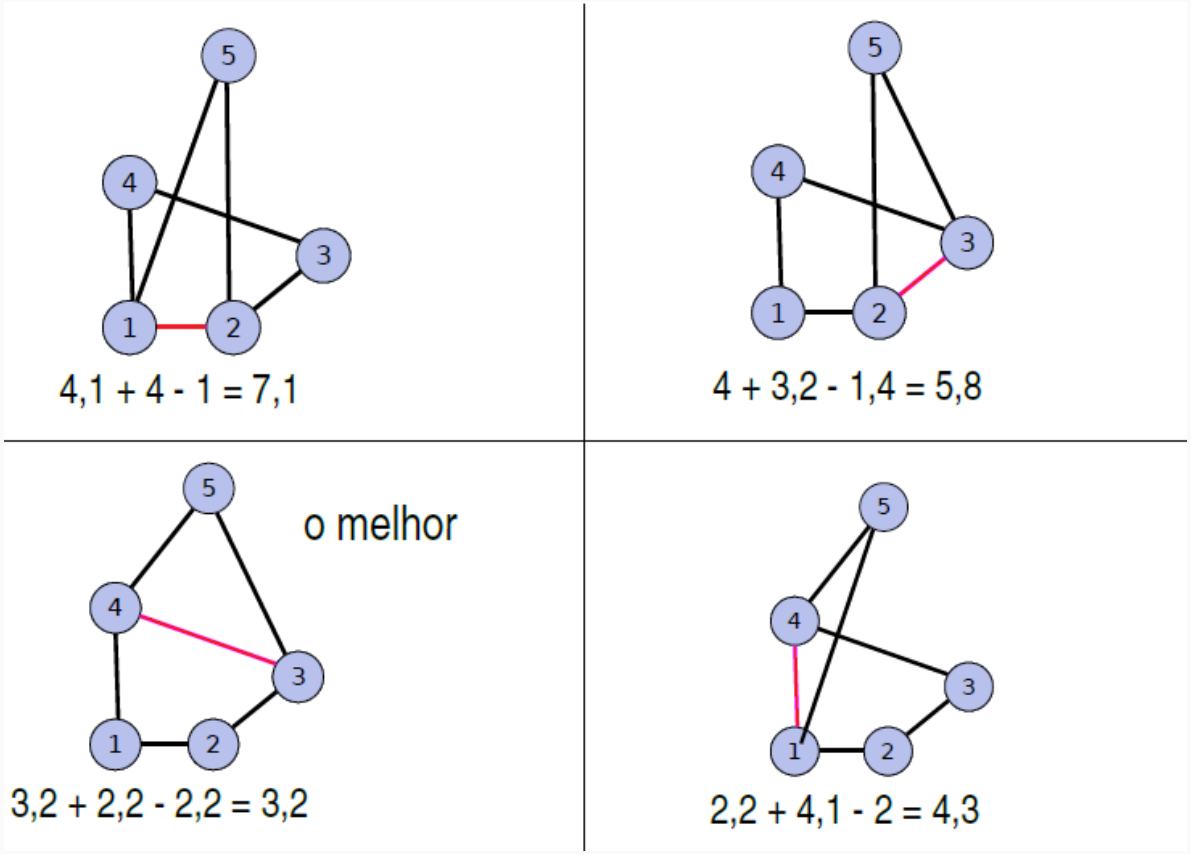
Encontrar k entre as cidades i e j tal que minimize

$s_{ij} = c_{ik} + c_{kj} - c_{ij}$



Algoritmo Guloso - PCV - inserção mais barata

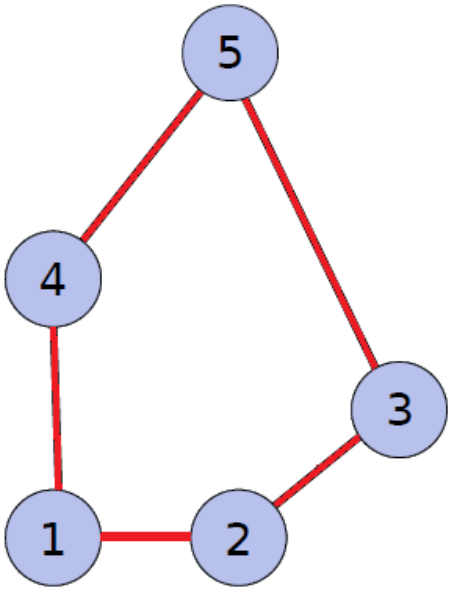
Encontrar k entre as cidades i e j tal que minimize $s_{ij} = c_{ik} + c_{kj} - c_{ij}$



Solução final

Com matriz de distâncias entre 5 cidades:

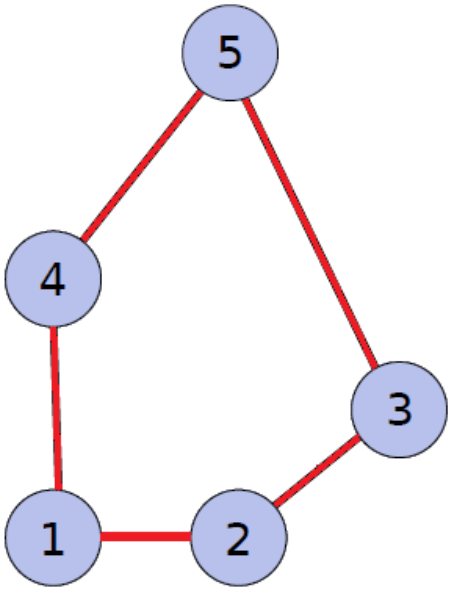
	1	**2**	**3**	**4**	**5**
1	0	1	2,2	2	4,1
2	1	0	1,4	2,2	4
3	2,2	1,4	0	2,2	3,2
4	2	2,2	2,2	0	2,2
5	4,1	4	3,2	2,2	0



Solução final

Com matriz de distâncias entre 5 cidades:

	1	**2**	**3**	**4**	**5**
1	0	1	2,2	2	4,1
2	1	0	1,4	2,2	4
3	2,2	1,4	0	2,2	3,2
4	2	2,2	2,2	0	2,2
5	4,1	4	3,2	2,2	0



Meta-heurísticas

- Métodos Construtivos
- Busca Local
- ILS, VNS e GRASP

Busca Local

Objetivo

Melhorar as soluções construídas: explorando de maneira interativa o espaço de busca em cada passo, de uma solução para uma solução "vizinha".

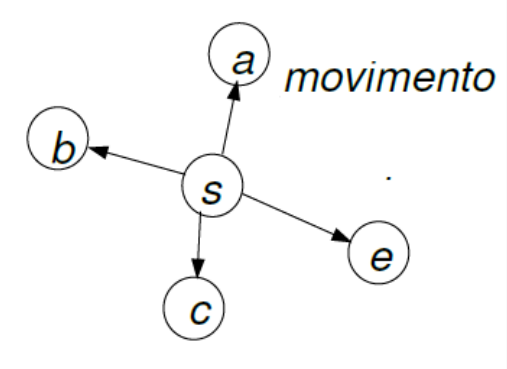
Métodos

- Método da Descida/Subida
- Método de Descida/Subida com Primeira Melhora
- Método de Descida/Subida Randômica
- Método de Descida em Vizinhança Variável- VND
- Método da Pesquisa Interativa - ILS

UFOP
Universidade Federal de Ouro Preto

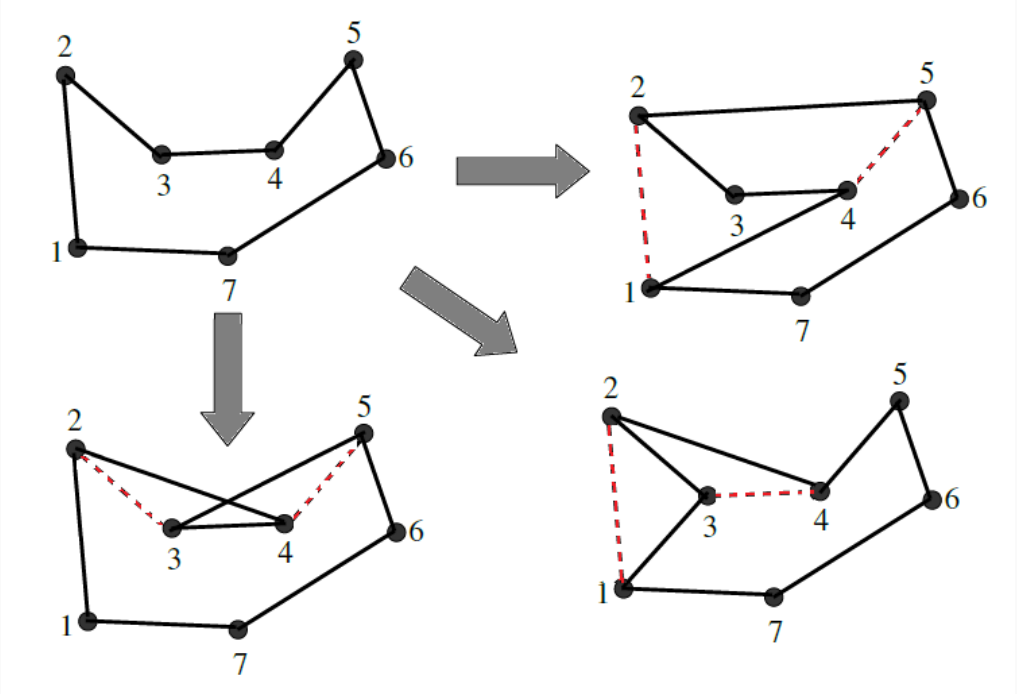
Noção de Vizinhança

- Seja S o espaço de busca do problema.
- $s \in S$ uma solução construída.
- Seja N uma função que associa a cada solução $s \in S$, um subconjunto $N(s) \subseteq S$, denominado a vizinhança de s.
- Cada solução $s' \in N(s)$ é chamada vizinho de s.
- Denomina-se movimento uma modificação que transforma uma solução s em outra solução, s', tal que $s' \in N(s)$. Exemplo: $N(s) = \{a, b, c, d\}$



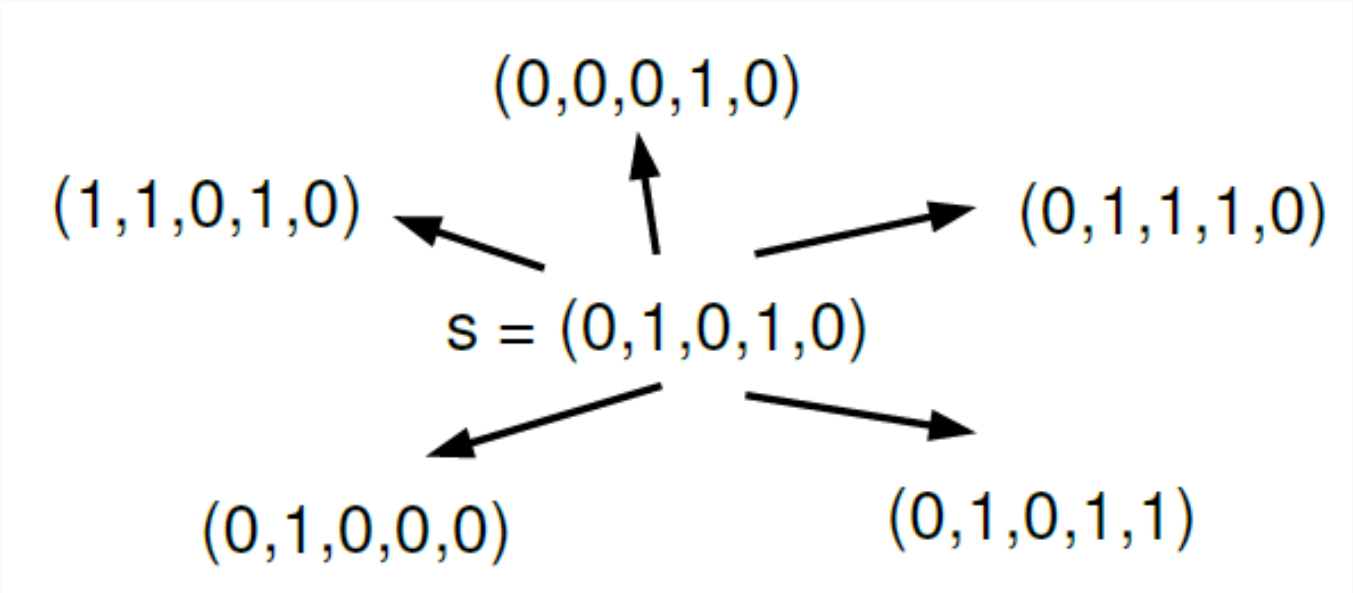
Problema do Caixeiro Viajante

Uma vizinhança $N(s)$ de uma solução s, definida pelo movimento de remover duas arestas e substituir por outras duas arestas (válidas). (2-opt)



Problema da mochila

Uma vizinhança $N(s)$ de uma solução s, é definida pelo movimento de mudar a variável de s de 0 para 1 ou vice-versa.



Método da descida - (*hill climbing*)

- Seleciona uma solução inicial $s \in S$.
- A cada passo analisa todos os possíveis vizinhos
- Move para o vizinho que apresenta uma melhora no valor atual da função de avaliação (objetivo)
- O método pára quando um ótimo local é encontrado, ou seja, nenhum vizinho for melhor do que a solução corrente (segundo aquele movimento).

Método da Descida

$s \leftarrow \text{Gera}()$
$k \leftarrow 1$
while k **do**
  $s' \leftarrow \textit{EscolheMelhorVizinho}(s, N)$
  if $f(s') < f(s)$ **then**
    $s \leftarrow s'$
  else
    $k \leftarrow 0$
  end-if
end-while
return s

Busca Local - O problema da mochila

$objeto(j)$	1	2	3	4	5	6	7	8
$valor(c_j)$	3	3	2	4	2	3	5	2
$peso(w_j)$	5	4	7	8	4	4	6	8

Capacidade da mochila b $= 25$

Busca Local

- Solução inicial aleatória.
- Uma vizinhança $N(s)$ de uma solução s, é definida pelo movimento de mudar a variável de s de 0 para 1 ou vice-versa.

Busca Local - O problema da mochila

$objeto(j)$	1	2	3	4	5	6	7	8
$valor(c_j)$	3	3	2	4	2	3	5	2
$peso(w_j)$	5	4	7	8	4	4	6	8

Capacidade da mochila b $= 25$

Busca Local

- Solução inicial aleatória.
- Uma vizinhança $N(s)$ de uma solução s, é definida pelo movimento de mudar a variável de s de 0 para 1 ou vice-versa.

Busca Local - O problema da mochila

Solução inicial: s = (10010000) - interação 1

Vizinho	*valor*	*peso*
00010000	4	8
11010000	10	17
10110000	9	20
10000000	3	5
10011000	9	17
10010100	10	17
10010010	12	19
10010001	9	21

Busca Local - O problema da mochila

Solução inicial: s = (10010010) - interação 2

Vizinho	*valor*	*peso*
00010010	9	14
11010010	15	23
10110010	14	26
10000010	8	11
10011010	14	23
10010110	15	23
10010000	7	13
10010011	14	27

Busca Local - O problema da mochila

Solução inicial: s = (11010010) - interação 3

Vizinho	*valor*	*peso*
01010010	12	18
10010010	12	19
11110010	17	30
11000010	11	15
11011010	17	27
11010110	18	27
11010000	10	17
11010011	17	31

Não é possível melhorar!.

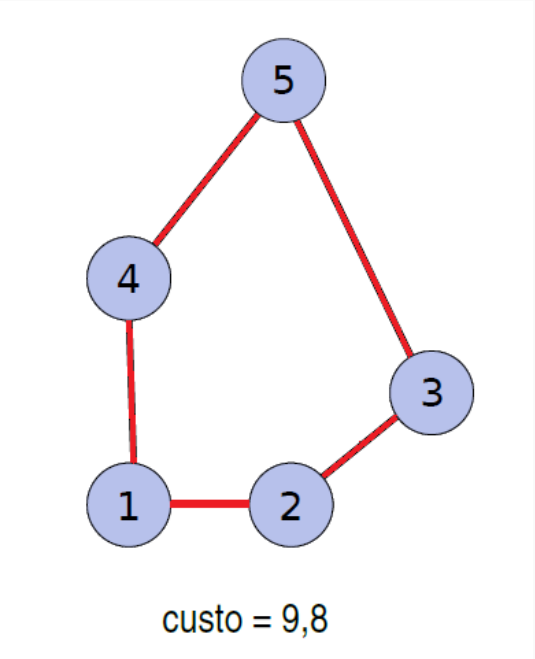
UFOP
Universidade Federal de Ouro Preto

Considere o PCV

Com matriz de distâncias entre 5 cidades:

	1	**2**	**3**	**4**	**5**
1	0	1	2,2	2	4,1
2	1	0	1,4	2,2	4
3	2,2	1,4	0	2,2	3,2
4	2	2,2	2,2	0	2,2
5	4,1	4	3,2	2,2	0

- Solução inicial → inserção mais barata.

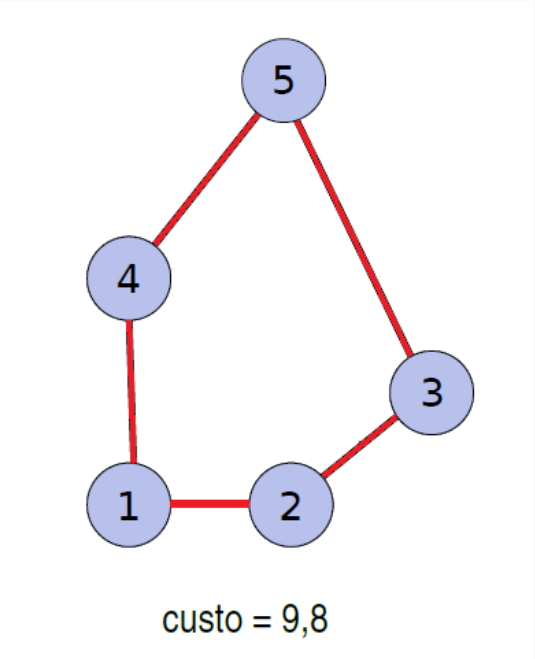


Considere o PCV

Com matriz de distâncias entre 5 cidades:

	1	**2**	**3**	**4**	**5**
1	0	1	2,2	2	4,1
2	1	0	1,4	2,2	4
3	2,2	1,4	0	2,2	3,2
4	2	2,2	2,2	0	2,2
5	4,1	4	3,2	2,2	0

- \- Solução inicial $\rightarrow$ inserção mais barata.

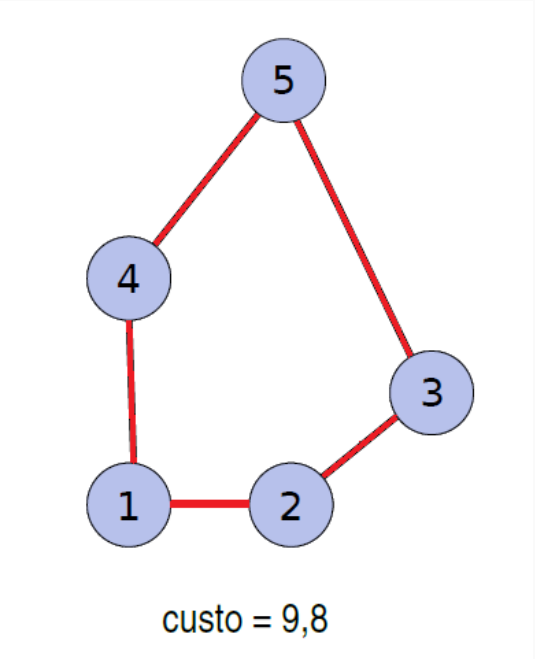


Considere o PCV

Com matriz de distâncias entre 5 cidades:

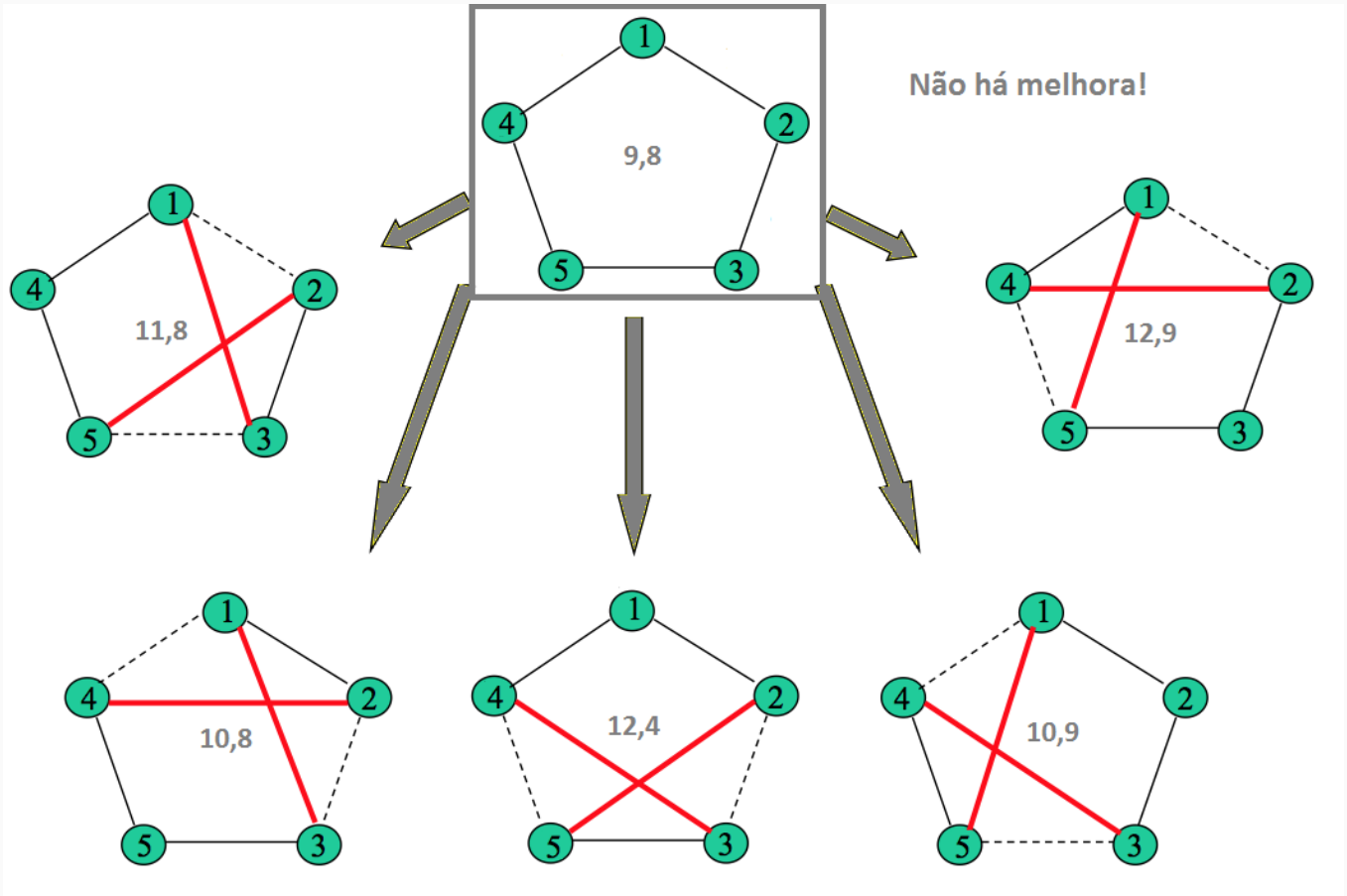
	1	**2**	**3**	**4**	**5**
1	0	1	2,2	2	4,1
2	1	0	1,4	2,2	4
3	2,2	1,4	0	2,2	3,2
4	2	2,2	2,2	0	2,2
5	4,1	4	3,2	2,2	0

- \- Solução inicial $\rightarrow$ inserção mais barata.



Busca Local - PCV (2-opt)

Solução inicial: s = (12354) → custo = 9,8



Variantes da Busca Local

- Há casos onde a vizinhança é grande ou o custo de avaliação de cada solução é muito caro.
- Alternativas para evitar a avaliação de toda vizinhança:
  - Método de Primeira Melhora
  - Método de Descida Aleatória

Método da Descida de Primeira Melhora

- Interrompe a exploração da vizinhança quando um primeiro vizinho melhor é encontrado
- Desta forma, apenas no pior caso, toda a vizinhança será explorada
- A solução final é um ótimo local com respeito à vizinhança considerada
- Evita a pesquisa exaustiva pelo melhor vizinho

Método da Descida de Primeira Melhora

$s \leftarrow \text{Gera}()$
$k \leftarrow 1$
while k **do**
  $s' \leftarrow \textit{EscolheVizinho}(s, N)$
  if $f(s') < f(s)$ **then**
    $s \leftarrow s'$
  end-if
  $k \leftarrow |(s, N)|$
end-while
return s

Método da Descida Aleatória

- Consiste em escolher um vizinho aleatório e o aceitar somente se ele for melhor do que a melhor solução conhecida até o momento
- Se o vizinho escolhido não for melhor, a solução corrente (melhor) permanece inalterada e outro vizinho é gerado
- O procedimento é interrompido após um certo número fixo de iterações sem melhora no valor da função objetivo
- A solução final não é necessariamente um ótimo local
- Por outro lado, é possível controlar melhor o tempo de processamento

Método da Descida Aleatótia

$s \leftarrow \text{Gera}()$
$k \leftarrow 1$
repeat
  $s' \leftarrow \textit{EscolheVizinhoAleatorio}(s, N)$
  if $f(s') < f(s)$ **then**
    $s \leftarrow s'$
    $k \leftarrow 1$
  else
    $k \leftarrow k + 1$
  end-if
until $k = max$
return s

Meta-heurísticas

- Métodos Construtivos
- Busca Local
- ILS, VNS e GRASP

UFOP
Universidade Federal de Ouro Preto

Algoritmos de ILS e VNS

- Pesquisa Local Iterativa (ILS) e Pesquisa de Vizinhança Variável (VNS) baseiam-se em pesquisa local.
- Empregam estratégias muito explícitas para "escapar" de ótimos locais: **Perturbação** (ILS) e **mudança de vizinhança** (VNS).
- Embora recentes, apresentam muitos casos de sucesso em problemas de otimização combinatória.

Algoritmos de ILS e VNS

- Pesquisa Local Iterativa (ILS) e Pesquisa de Vizinhança Variável (VNS) baseiam-se em pesquisa local.
- Empregam estratégias muito explícitas para "escapar" de ótimos locais: **Perturbação** (ILS) e **mudança de vizinhança** (VNS).
- Embora recentes, apresentam muitos casos de sucesso em problemas de otimização combinatória.

UFOP
Universidade Federal de Ouro Preto

Algoritmos de ILS e VNS

- Pesquisa Local Iterativa (ILS) e Pesquisa de Vizinhança Variável (VNS) baseiam-se em pesquisa local.
- Empregam estratégias muito explícitas para "escapar" de ótimos locais: **Perturbação** (ILS) e **mudança de vizinhança** (VNS).
- Embora recentes, apresentam muitos casos de sucesso em problemas de otimização combinatória.

UFOP
Universidade Federal de Ouro Preto

Pesquisa Local Iterativa (ILS)

- Para um problema P de otimização combinatória com uma função de custo f, seja S o conjunto de soluções admissíveis.
- Para uma dada **vizinhança** N, seja S^* um conjunto de **ótimos locais** em P e $S^* \subseteq S$.
- Um algoritmo **PesquisaLocal** que define o mapeamento de S para S^*.

Pesquisa Local Iterativa (ILS)

- Para um problema P de otimização combinatória com uma função de custo f, seja S o conjunto de soluções admissíveis.
- Para uma dada **vizinhança** N, seja S^* um conjunto de **ótimos locais** em P e $S^* \subseteq S$.
- Um algoritmo **PesquisaLocal** que define o mapeamento de S para S^*.

Pesquisa Local Iterativa (ILS)

- Para um problema P de otimização combinatória com uma função de custo f, seja S o conjunto de soluções admissíveis.
- Para uma dada **vizinhança** N, seja S^* um conjunto de **ótimos locais** em P e $S^* \subseteq S$.
- Um algoritmo **PesquisaLocal** que define o mapeamento de S para S^*.

Pesquisa Local Iterativa (ILS)

- ILS efetua uma amostragem independente (enviesada) em S^*:
  1. Dada uma solução $s^* \in S^*$, **perturba-a** para gerar outra solução $s' \in S$;
  2. Aplica **PesquisaLocal** a partir de s' para gerar uma solução $s^{*'} \in S^*$;
  3. **Aceita** s^* **ou** $s^{*'}$
- ILS é um método de *pesquisa local* em S^* (mas sem ter uma noção explícita de vizinhança).

Pesquisa Local Iterativa (ILS)

- ILS efetua uma amostragem independente (enviesada) em S^*:
  1. Dada uma solução $s^* \in S^*$, **perturba-a** para gerar outra solução $s' \in S$;
  2. Aplica **PesquisaLocal** a partir de s' para gerar uma solução $s^{*'} \in S^*$;
  3. **Aceita** s^* **ou** $s^{*'}$
- ILS é um método de *pesquisa local* em S^* (mas sem ter uma noção explícita de vizinhança).

Pesquisa Local Iterativa (ILS)

- ILS efetua uma amostragem independente (enviesada) em S^*:
  1. Dada uma solução $s^* \in S^*$, **perturba-a** para gerar outra solução $s' \in S$;
  2. Aplica **PesquisaLocal** a partir de s' para gerar uma solução $s^{*'} \in S^*$;
  3. **Aceita** s^* **ou** $s^{*'}$
- ILS é um método de *pesquisa local* em S^* (mas sem ter uma noção explícita de vizinhança).

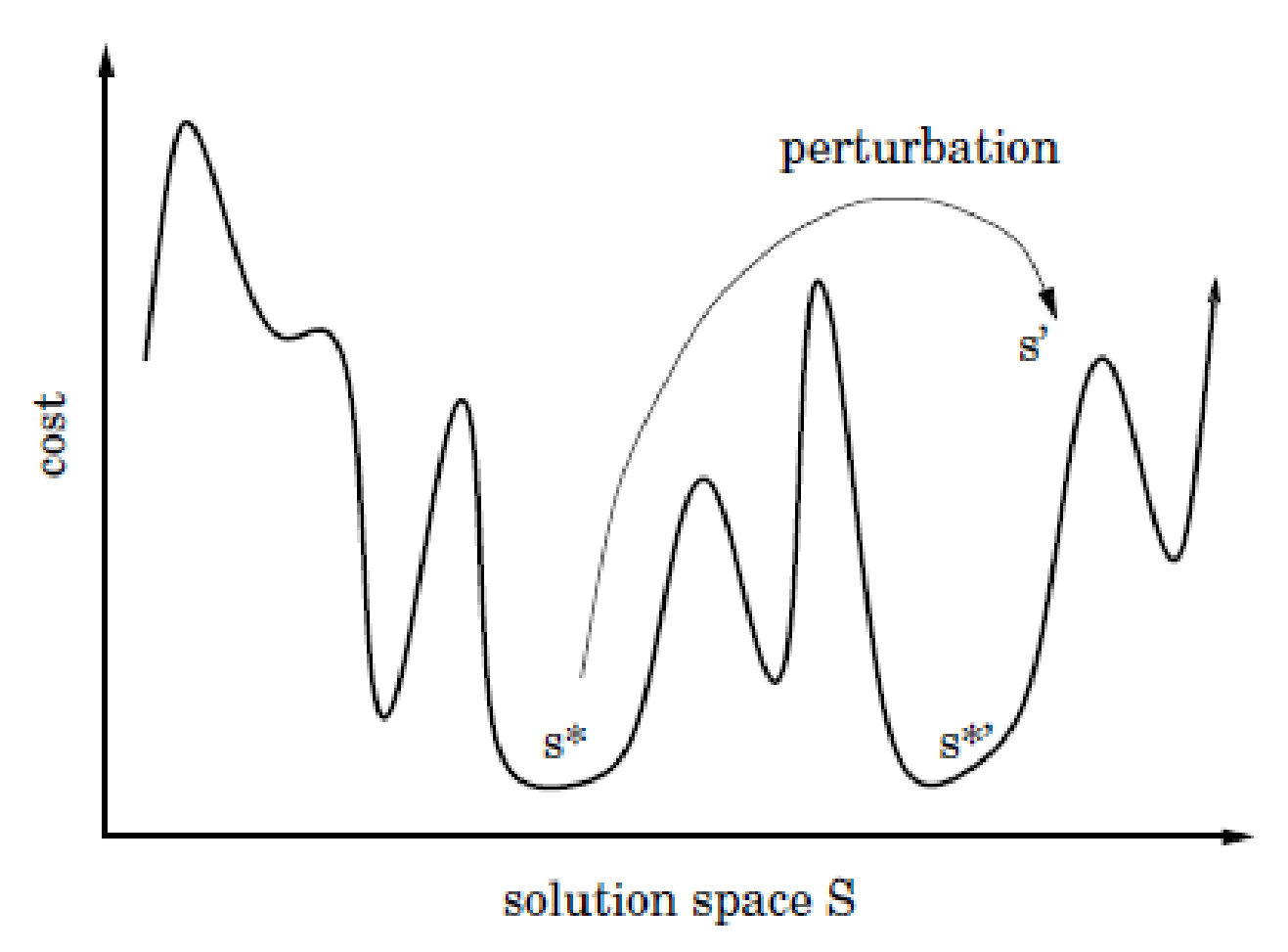
Pesquisa Local Iterativa (ILS)

- ILS efetua uma amostragem independente (enviesada) em S^*:
  1. Dada uma solução $s^* \in S^*$, **perturba-a** para gerar outra solução $s' \in S$;
  2. Aplica **PesquisaLocal** a partir de s' para gerar uma solução $s^{*'} \in S^*$;
  3. **Aceita** s^* **ou** $s^{*'}$
- ILS é um método de *pesquisa local* em S^* (mas sem ter uma noção explícita de vizinhança).

Pesquisa Local Iterativa (ILS)

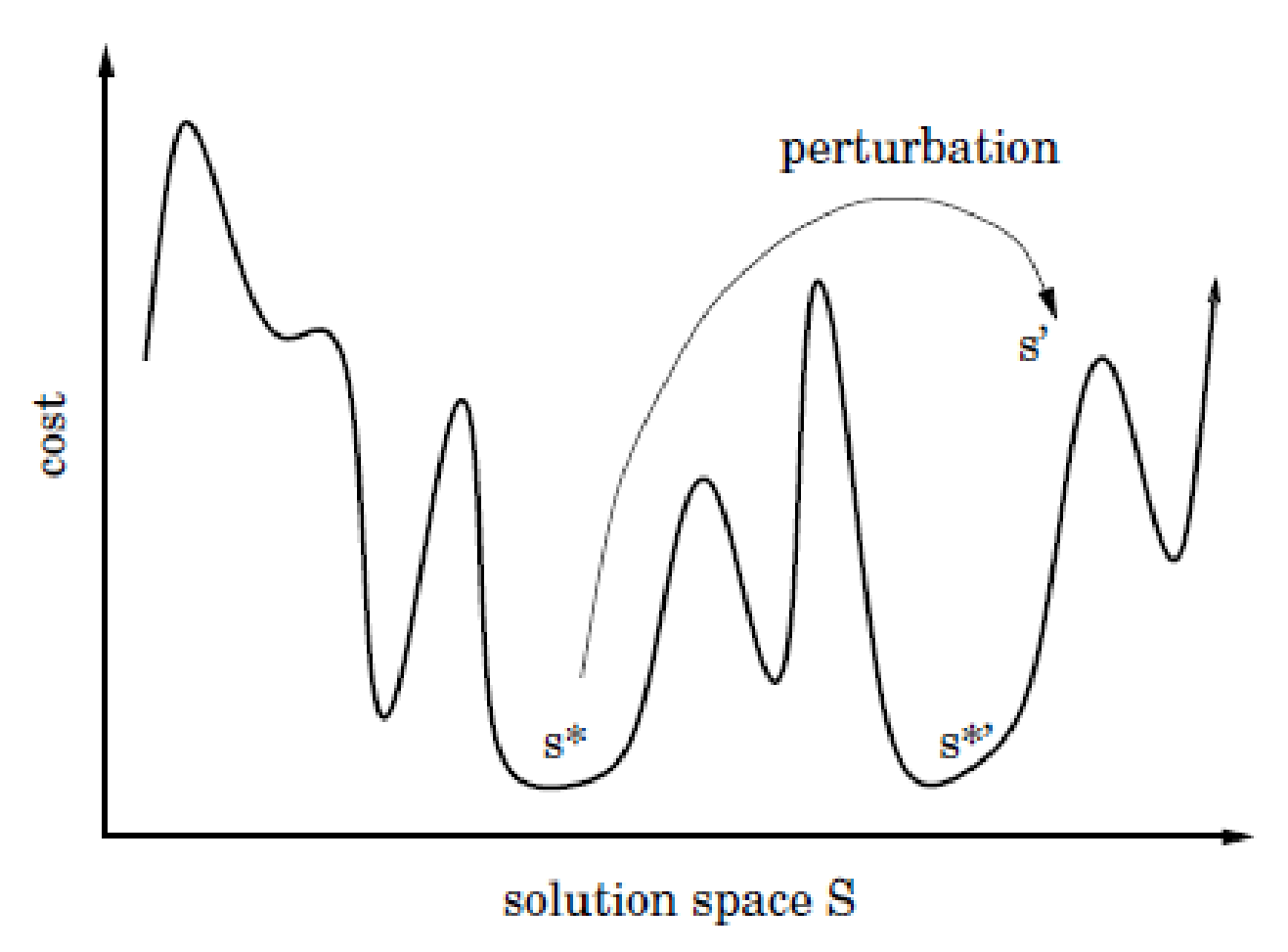
- ILS efetua uma amostragem independente (enviesada) em S^*:
  1. Dada uma solução $s^* \in S^*$, **perturba-a** para gerar outra solução $s' \in S$;
  2. Aplica **PesquisaLocal** a partir de s' para gerar uma solução $s^{*'} \in S^*$;
  3. **Aceita** s^* **ou** $s^{*'}$
- ILS é um método de *pesquisa local* em S^* (mas sem ter uma noção explícita de vizinhança).

Pesquisa Local Iterativa (ILS)



Perturbação permite saltar entre bacias de atração.

Pesquisa Local Iterativa (ILS)



Perturbação permite saltar entre bacias de atração.

Pesquisa Local Iterativa (ILS)

$s \leftarrow$ Gera()
$s^* \leftarrow$ *PesquisaLocal*(s)
repeat
  $s' \leftarrow$ *Perturba*(s^*, memória)
  $s^{*'} \leftarrow$ *PesquisaLocal*(s')
  $s^* \leftarrow$ *Aceita*($s^*, s^{*'}$)
until condição de parada ser verdadeira
return s^*

Pesquisa Local Iterativa (ILS)

- Versão básica de ILS::
  - Gera: Aleatório ou heurística;
  - PesquisaLocal: Muitas vezes disponível na literatura;
  - Perturba: Movimento aleatório numa vizinhança maior;
  - Aceita: Aceita o ótimo local com menor custo
- Versão básica é muitas vezes suficiente.
- Requer poucas linhas de código.

Pesquisa Local Iterativa (ILS)

- Versão básica de ILS::
  - Gera: Aleatório ou heurística;
  - PesquisaLocal: Muitas vezes disponível na literatura;
  - Perturba: Movimento aleatório numa vizinhança maior;
  - Aceita: Aceita o ótimo local com menor custo
- Versão básica é muitas vezes suficiente.
- Requer poucas linhas de código.

Pesquisa Local Iterativa (ILS)

- Versão básica de ILS::
  - Gera: Aleatório ou heurística;
  - PesquisaLocal: Muitas vezes disponível na literatura;
  - Perturba: Movimento aleatório numa vizinhança maior;
  - Aceita: Aceita o ótimo local com menor custo
- Versão básica é muitas vezes suficiente.
- Requer poucas linhas de código.

Pesquisa Local Iterativa (ILS)

- Versão básica de ILS::
  - Gera: Aleatório ou heurística;
  - PesquisaLocal: Muitas vezes disponível na literatura;
  - Perturba: Movimento aleatório numa vizinhança maior;
  - Aceita: Aceita o ótimo local com menor custo
- Versão básica é muitas vezes suficiente.
- Requer poucas linhas de código.

Pesquisa Local Iterativa (ILS)

- Versão básica de ILS::
  - Gera: Aleatório ou heurística;
  - PesquisaLocal: Muitas vezes disponível na literatura;
  - Perturba: Movimento aleatório numa vizinhança maior;
  - Aceita: Aceita o ótimo local com menor custo
- Versão básica é muitas vezes suficiente.
- Requer poucas linhas de código.

Pesquisa Local Iterativa (ILS)

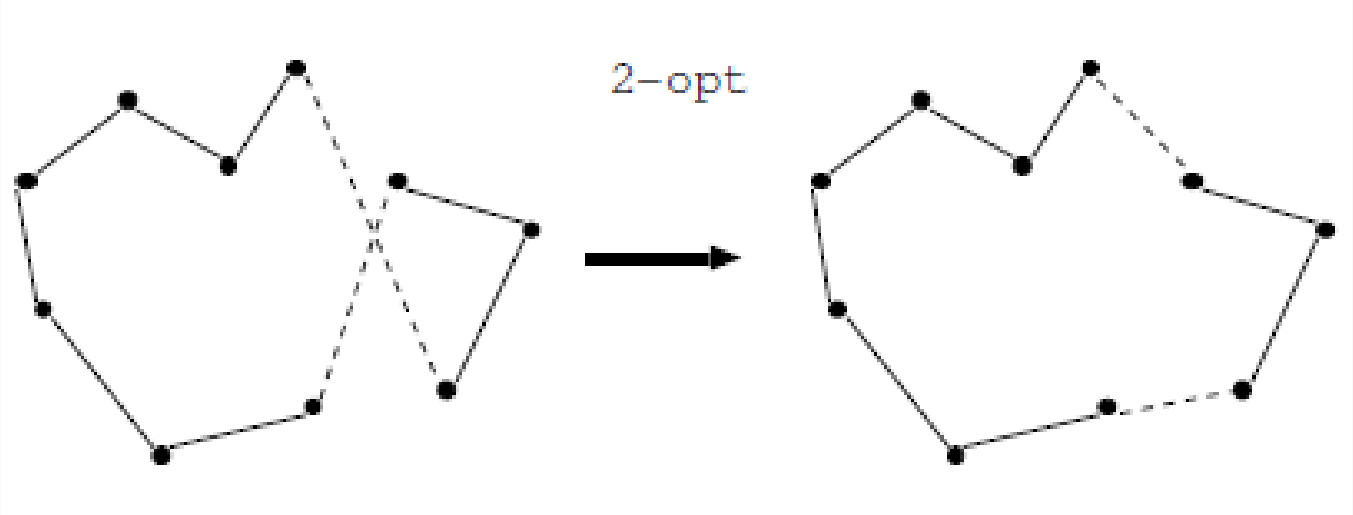
- Versão básica de ILS::
  - Gera: Aleatório ou heurística;
  - PesquisaLocal: Muitas vezes disponível na literatura;
  - Perturba: Movimento aleatório numa vizinhança maior;
  - Aceita: Aceita o ótimo local com menor custo
- Versão básica é muitas vezes suficiente.
- Requer poucas linhas de código.

Pesquisa Local Iterativa (ILS)

- Versão básica de ILS::
  - Gera: Aleatório ou heurística;
  - PesquisaLocal: Muitas vezes disponível na literatura;
  - Perturba: Movimento aleatório numa vizinhança maior;
  - Aceita: Aceita o ótimo local com menor custo
- Versão básica é muitas vezes suficiente.
- Requer poucas linhas de código.

ILS para o Problema do Caixeiro Viajante

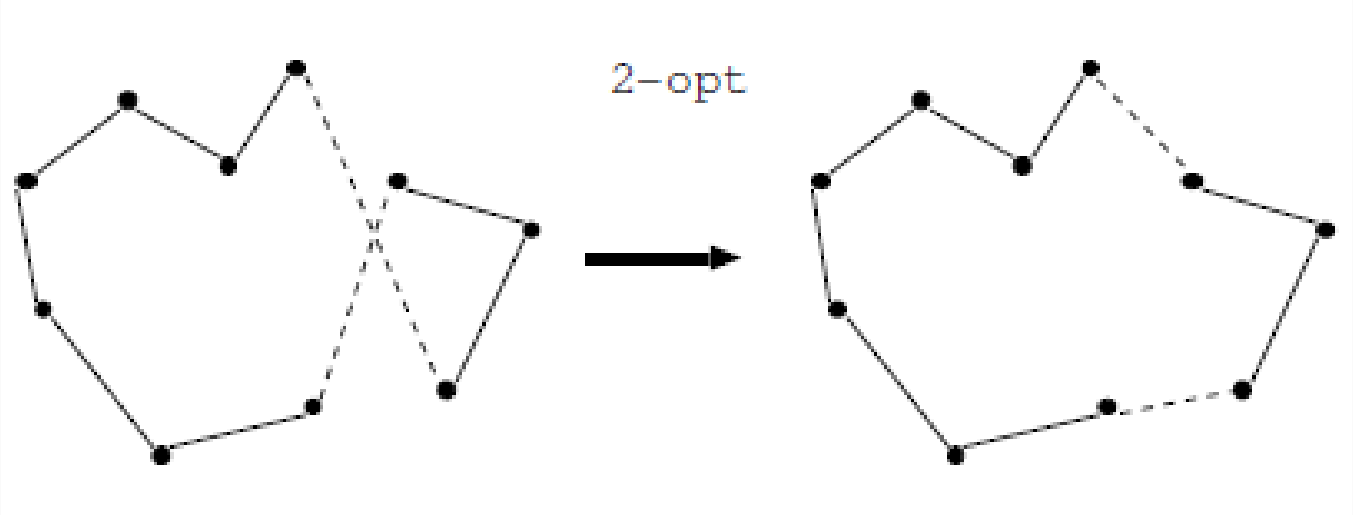
- Gera: Heurística gulosa;
- PesquisaLocal: 2-opt ou 3-opt;



- Perturba: Double-bridge (4-opt);
- Aceita: Aceita a solução com menor custo.

ILS para o Problema do Caixeiro Viajante

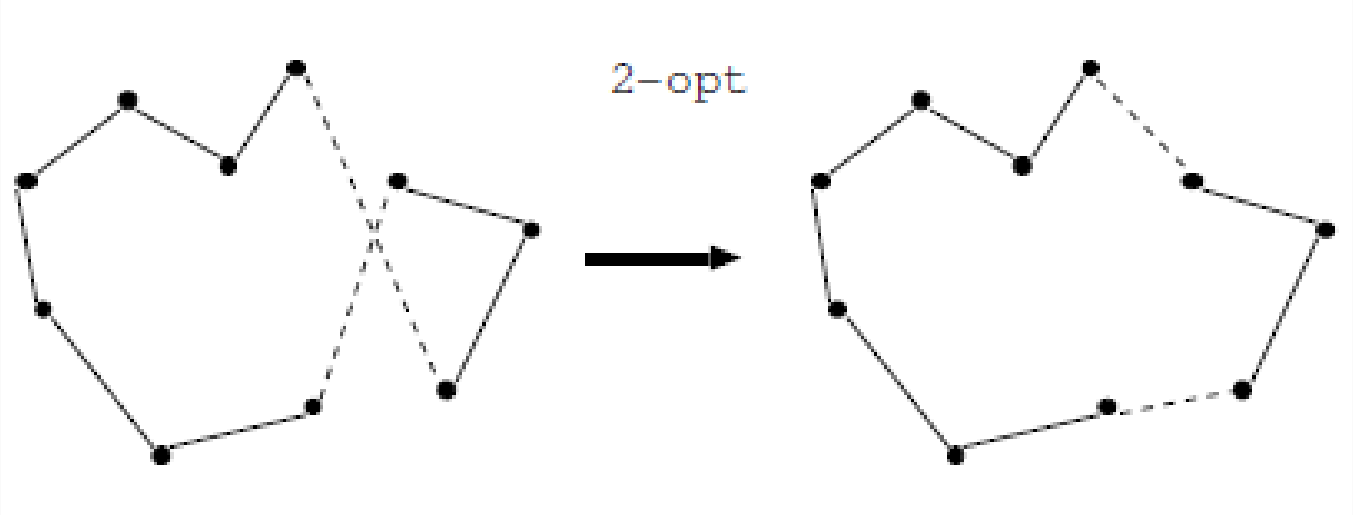
- Gera: Heurística gulosa;
- PesquisaLocal: 2-opt ou 3-opt;



- Perturba: Double-bridge (4-opt);
- Aceita: Aceita a solução com menor custo.

ILS para o Problema do Caixeiro Viajante

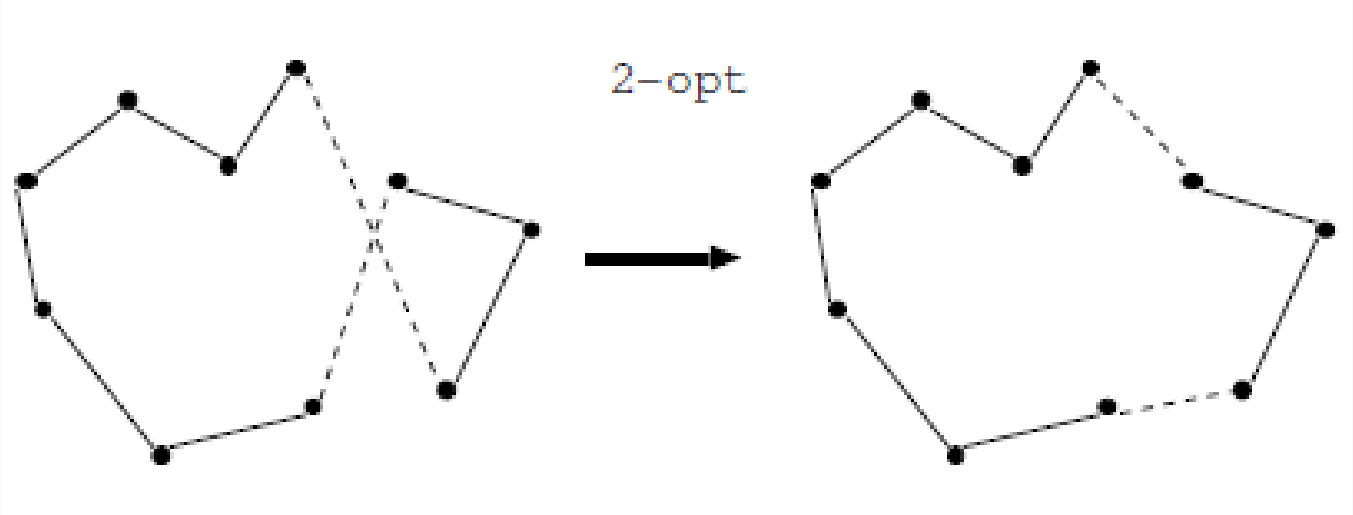
- Gera: Heurística gulosa;
- PesquisaLocal: 2-opt ou 3-opt;



- Perturba: Double-bridge (4-opt);
- Aceita: Aceita a solução com menor custo.

ILS para o Problema do Caixeiro Viajante

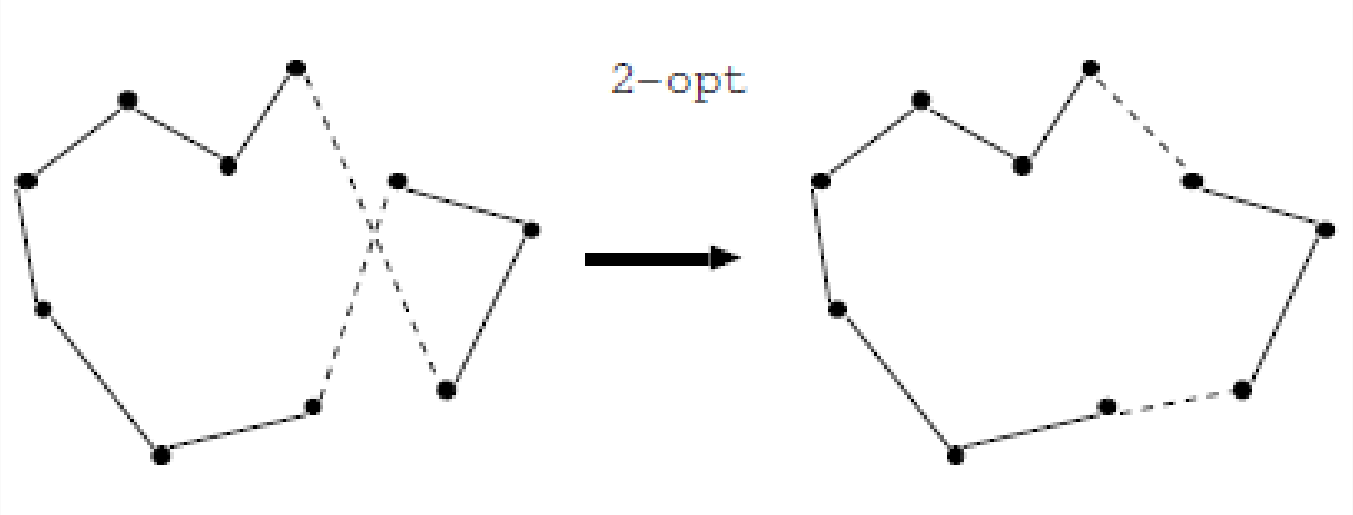
- Gera: Heurística gulosa;
- PesquisaLocal: 2-opt ou 3-opt;



- Perturba: Double-bridge (4-opt);
- Aceita: Aceita a solução com menor custo.

ILS para o Problema do Caixeiro Viajante

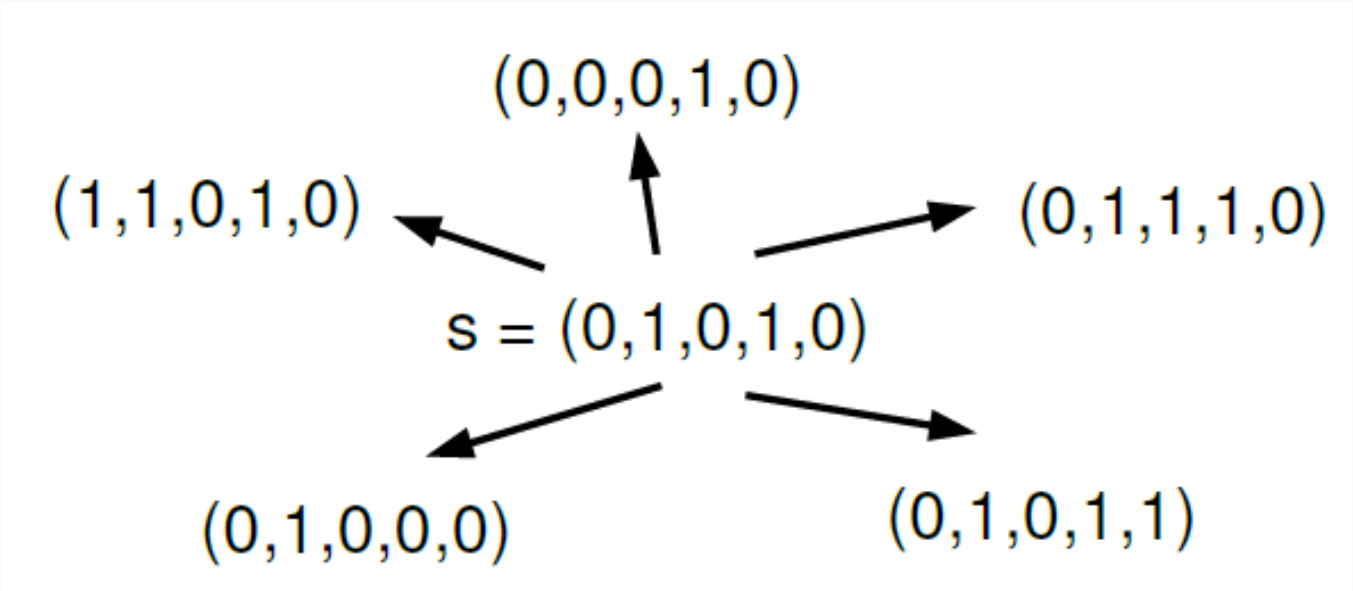
- Gera: Heurística gulosa;
- PesquisaLocal: 2-opt ou 3-opt;



- Perturba: Double-bridge (4-opt);
- Aceita: Aceita a solução com menor custo.

ILS para o Problema da Mochila

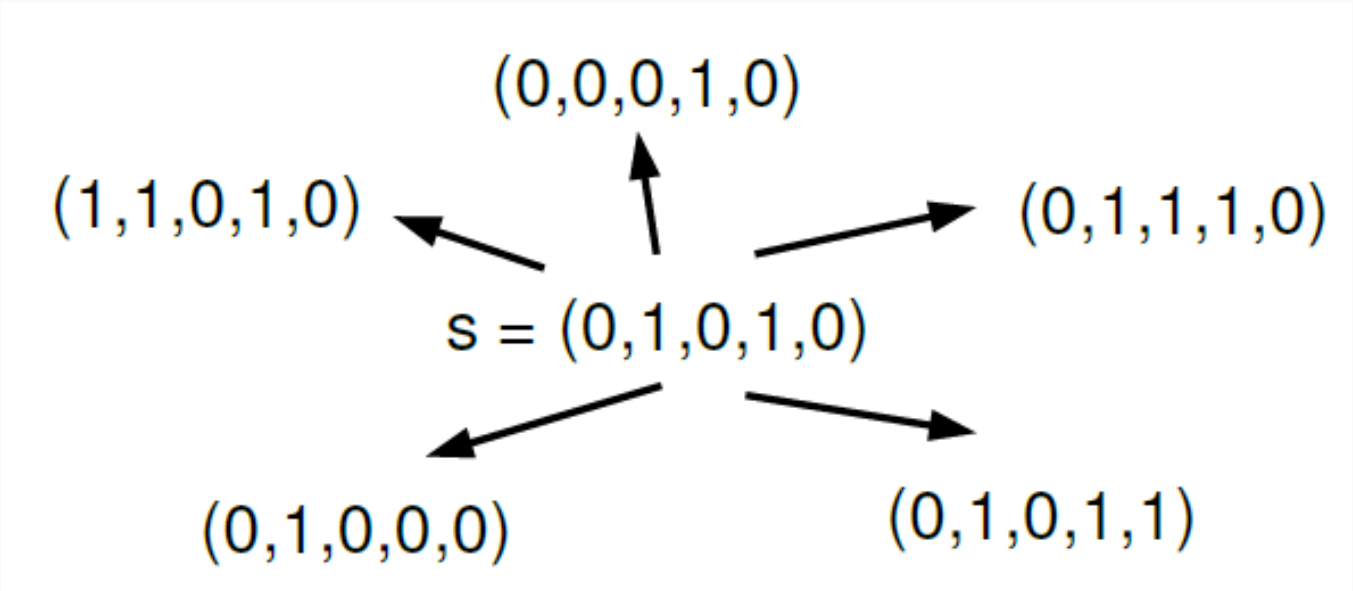
- Gera: Heurística gulosa;
- PesquisaLocal: Inversão das variáveis $0, 1$;



- Perturba: Complementar k variáveis da solução corrente;
- Aceita: Aceita a solução com maior custo.

ILS para o Problema da Mochila

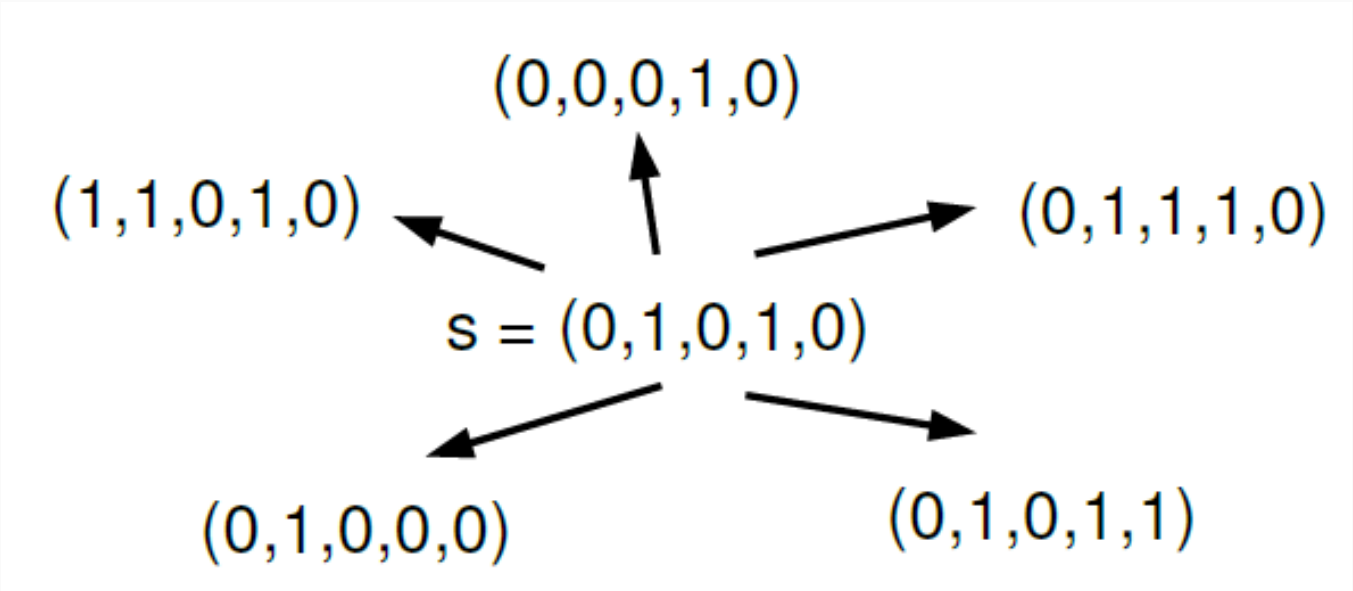
- Gera: Heurística gulosa;
- PesquisaLocal: Inversão das variáveis $0, 1$;



- Perturba: Complementar k variáveis da solução corrente;
- Aceita: Aceita a solução com maior custo.

ILS para o Problema da Mochila

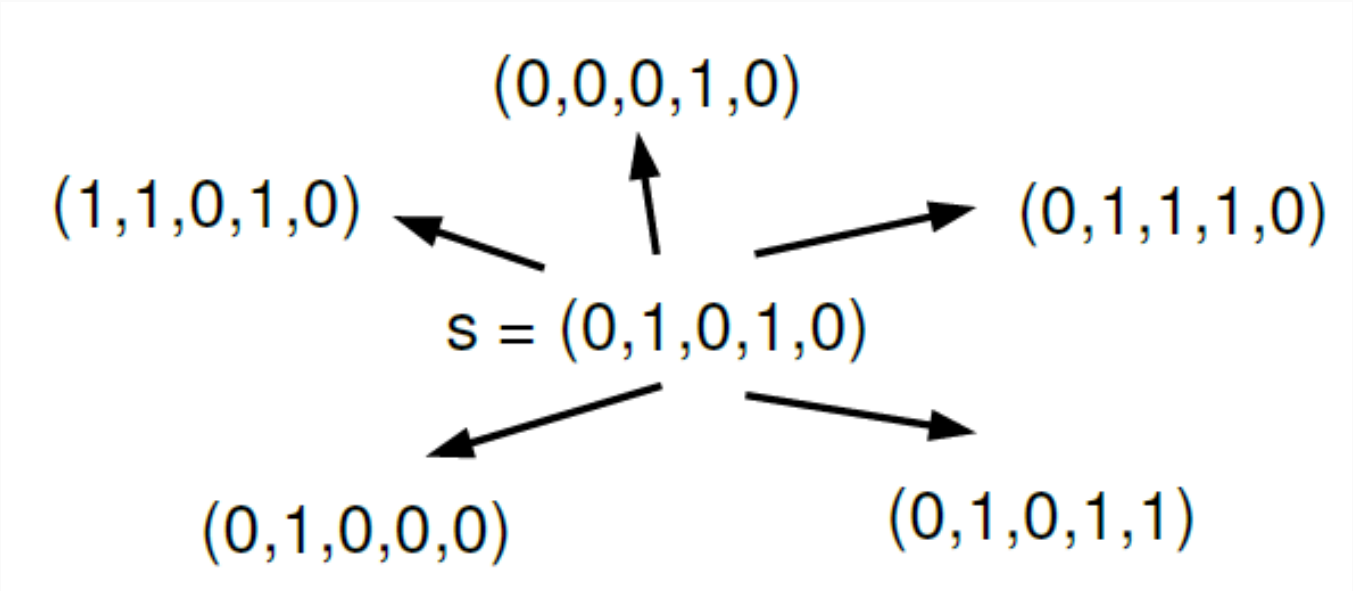
- Gera: Heurística gulosa;
- PesquisaLocal: Inversão das variáveis $0, 1$;



- Perturba: Complementar k variáveis da solução corrente;
- Aceita: Aceita a solução com maior custo.

ILS para o Problema da Mochila

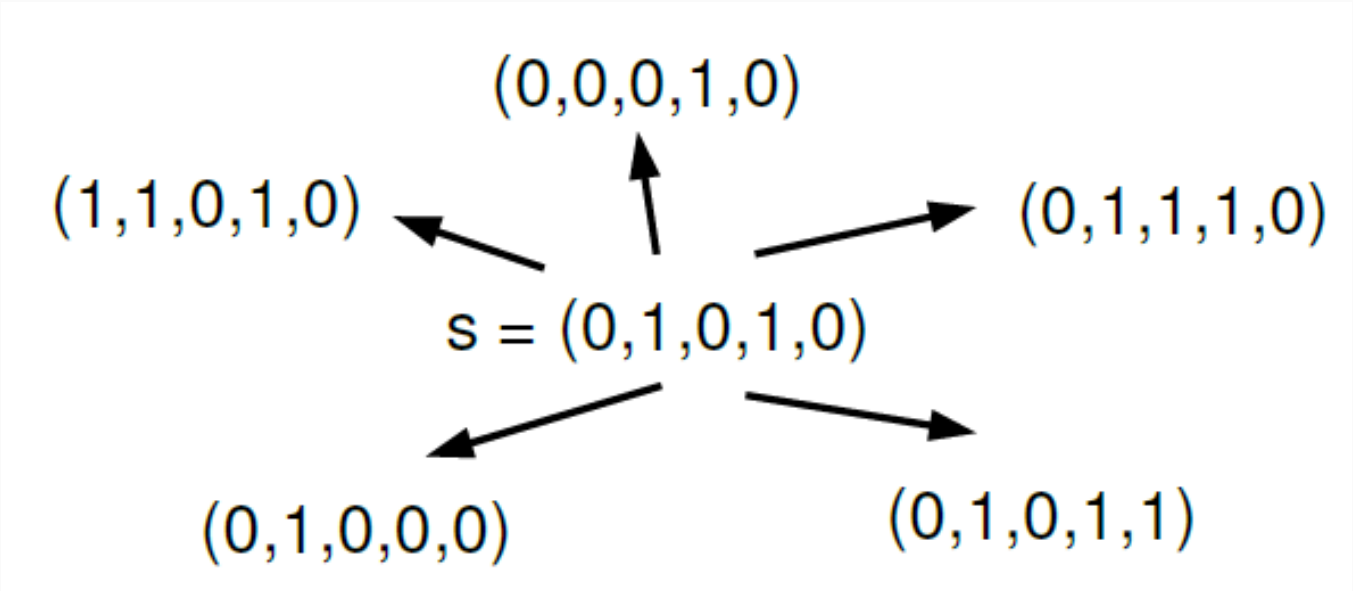
- Gera: Heurística gulosa;
- PesquisaLocal: Inversão das variáveis $0, 1$;



- Perturba: Complementar k variáveis da solução corrente;
- Aceita: Aceita a solução com maior custo.

ILS para o Problema da Mochila

- Gera: Heurística gulosa;
- PesquisaLocal: Inversão das variáveis $0, 1$;



- Perturba: Complementar k variáveis da solução corrente;
- Aceita: Aceita a solução com maior custo.

Pesquisa de Vizinhança Variável (VNS)

"VNS é uma meta-heurística que é baseada na mudança sistemática da vizinhança."

- Um ótimo local na vizinhança N não é necessariamente um ótimo local para outra vizinhança N'.

Pesquisa de Vizinhança Variável (VNS)

"VNS é uma meta-heurística que é baseada na mudança sistemática da vizinhança."

- Um ótimo local na vizinhança N não é necessariamente um ótimo local para outra vizinhança N'.

Pesquisa de Vizinhança Variável (VNS)

- O princípio de VNS é modificar a vizinhança durante a pesquisa.
- Assume-se a existência de vizinhanças $(N^1, \ldots, N^m)$.
- Diversas adaptações deste princípio:
  - Variable neighborhood descent (VND)
  - Basic variable neighborhood search
  - Reduced variable neighborhoood search

Pesquisa de Vizinhança Variável (VNS)

- O princípio de VNS é modificar a vizinhança durante a pesquisa.
- Assume-se a existência de vizinhanças ($N^1, \ldots, N^m$).
- Diversas adaptações deste princípio:
  - Variable neighborhood descent (VND)
  - Basic variable neighborhood search
  - Reduced variable neighborhoood search

Pesquisa de Vizinhança Variável (VNS)

- O princípio de VNS é modificar a vizinhança durante a pesquisa.
- Assume-se a existência de vizinhanças $(N^1, \ldots, N^m)$.
- Diversas adaptações deste princípio:
  - Variable neighborhood descent (VND)
  - Basic variable neighborhood search
  - Reduced variable neighborhoood search

Pesquisa de Vizinhança Variável (VNS)

- O princípio de VNS é modificar a vizinhança durante a pesquisa.
- Assume-se a existência de vizinhanças $(N^1, \ldots, N^m)$.
- Diversas adaptações deste princípio:
  - Variable neighborhood descent (VND)
  - Basic variable neighborhood search
  - Reduced variable neighborhoood search

Pesquisa de Vizinhança Variável (VNS)

- O princípio de VNS é modificar a vizinhança durante a pesquisa.
- Assume-se a existência de vizinhanças $(N^1, \ldots, N^m)$.
- Diversas adaptações deste princípio:
  - Variable neighborhood descent (VND)
  - Basic variable neighborhood search
  - Reduced variable neighborhoood search

Pesquisa de Vizinhança Variável (VNS)

- O princípio de VNS é modificar a vizinhança durante a pesquisa.
- Assume-se a existência de vizinhanças $(N^1, \ldots, N^m)$.
- Diversas adaptações deste princípio:
  - Variable neighborhood descent (VND)
  - Basic variable neighborhood search
  - Reduced variable neighborhoood search

Pesquisa de Vizinhança Variável (VNS)

- O princípio de VNS é modificar a vizinhança durante a pesquisa.
- Assume-se a existência de vizinhanças $(N^1, \ldots, N^m)$.
- Diversas adaptações deste princípio:
  - Variable neighborhood descent (VND)
  - Basic variable neighborhood search
  - Reduced variable neighborhoood search

Variable Neighborhood Descent (VND)

$s \leftarrow$ Gera()
$k \leftarrow 1$
repeat
  $s' \leftarrow$ *EscolheMelhorVizinho*(s, N^k)
  if $f(s') < f(s)$ **then**
    $s \leftarrow s'$
    $k \leftarrow 1$
  else
    $k \leftarrow k + 1$
  end-if
until $k = m$
return s

Pesquisa de Vizinhança Variável (VNS)

Reduced VNS

$s \leftarrow$ Gera()
$k \leftarrow 1$
repeat
  $s' \leftarrow$ *EscolheQualquerVizinho*(s, N^k)
  if $f(s') < f(s)$ **then**
    $s \leftarrow s'$
    $k \leftarrow 1$
  else
    $k \leftarrow k + 1$
  end-if
until $k = m$ (ou critério de parada satisfeito)
return s

Pesquisa de Vizinhança Variável (VNS)

Basic VNS

$s \leftarrow$ Gera()
$k \leftarrow 1$
repeat
  $s' \leftarrow$ *EscolheQualquerVizinho*(s, N^k)
  $s^* \leftarrow$ *PesquisaLocal*(s')
  if $f(s^*) < f(s)$ **then**
    $s \leftarrow s^*$
    $k \leftarrow 1$
  else
    $k \leftarrow k + 1$
  end-if
until $k = m$ (ou critério de parada satisfeito)
return s

GRASP - *Greedy Randomized Adaptive Search Procedure*

- É uma metaheurística constituída por heurísticas construtivas e busca local. Consiste de múltiplas aplicações de busca local, cada uma iniciando de uma solução diferente. As soluções iniciais são geradas por algum tipo de construção randômica gulosa ou algum esquema de perturbação.
- Várias aplicações para encontrar soluções aproximadas para problemas de otimização combinatória.

GRASP - *Greedy Randomized Adaptive Search Procedure*

- É uma metaheurística constituída por heurísticas construtivas e busca local. Consiste de múltiplas aplicações de busca local, cada uma iniciando de uma solução diferente. As soluções iniciais são geradas por algum tipo de construção randômica gulosa ou algum esquema de perturbação.
- Várias aplicações para encontrar soluções aproximadas para problemas de otimização combinatória.

GRASP - *Greedy Randomized Adaptive Search Procedure*

$k \leftarrow 1$
repeat
  $s \leftarrow$ Construção(solução)
  $s' \leftarrow$ *PesquisaLocal*(s)
  $s^* \leftarrow$ *Aceita*(s, s')
until k=max (condição de parada ser verdadeira)
return s^*

Princípio:

Combinação de um método construtivo com busca local, em um procedimento iterativo com iterações completamente independentes

GRASP - *Greedy Randomized Adaptive Search Procedure*

$k \leftarrow 1$
repeat
  $s \leftarrow$ Construção(solução)
  $s' \leftarrow$ *PesquisaLocal*(s)
  $s^* \leftarrow$ *Aceita*(s, s')
until k=max (condição de parada ser verdadeira)
return s^*

Princípio:

Combinação de um método construtivo com busca local, em um procedimento iterativo com iterações completamente independentes

Construção (GRASP)

- Cada iteração da fase de construção:
  - usando uma função gulosa, avaliar o benefício de cada elemento
  - criar uma lista restrita de candidatos (LRC), formada pelos elementos de melhor avaliação
  - selecionar aleatoriamente um elemento da lista restrita de candidatos
  - adaptar a função gulosa com base na decisão do elemento que foi incluído
- Seleção aleatória é feita entre os melhores elementos da lista de candidatos (não necessariamente o melhor, como na escolha puramente gulosa).
- Diversificação baseada em aleatorização controlada: diferentes soluções construídas em diferentes iterações GRASP

Construção (GRASP)

- Cada iteração da fase de construção:
  - usando uma função gulosa, avaliar o benefício de cada elemento
  - criar uma lista restrita de candidatos (LRC), formada pelos elementos de melhor avaliação
  - selecionar aleatoriamente um elemento da lista restrita de candidatos
  - adaptar a função gulosa com base na decisão do elemento que foi incluído
- Seleção aleatória é feita entre os melhores elementos da lista de candidatos (não necessariamente o melhor, como na escolha puramente gulosa).
- Diversificação baseada em aleatorização controlada: diferentes soluções construídas em diferentes iterações GRASP

Construção (GRASP)

- Cada iteração da fase de construção:
  - usando uma função gulosa, avaliar o benefício de cada elemento
  - criar uma lista restrita de candidatos (LRC), formada pelos elementos de melhor avaliação
  - selecionar aleatoriamente um elemento da lista restrita de candidatos
  - adaptar a função gulosa com base na decisão do elemento que foi incluído
- Seleção aleatória é feita entre os melhores elementos da lista de candidatos (não necessariamente o melhor, como na escolha puramente gulosa).
- Diversificação baseada em aleatorização controlada: diferentes soluções construídas em diferentes iterações GRASP

Construção (GRASP)

- Cada iteração da fase de construção:
  - usando uma função gulosa, avaliar o benefício de cada elemento
  - criar uma lista restrita de candidatos (LRC), formada pelos elementos de melhor avaliação
  - selecionar aleatoriamente um elemento da lista restrita de candidatos
  - adaptar a função gulosa com base na decisão do elemento que foi incluído
- Seleção aleatória é feita entre os melhores elementos da lista de candidatos (não necessariamente o melhor, como na escolha puramente gulosa).
- Diversificação baseada em aleatorização controlada: diferentes soluções construídas em diferentes iterações GRASP

Construção (GRASP)

- Cada iteração da fase de construção:
  - usando uma função gulosa, avaliar o benefício de cada elemento
  - criar uma lista restrita de candidatos (LRC), formada pelos elementos de melhor avaliação
  - selecionar aleatoriamente um elemento da lista restrita de candidatos
  - adaptar a função gulosa com base na decisão do elemento que foi incluído
- Seleção aleatória é feita entre os melhores elementos da lista de candidatos (não necessariamente o melhor, como na escolha puramente gulosa).
- Diversificação baseada em aleatorização controlada: diferentes soluções construídas em diferentes iterações GRASP

Construção (GRASP)

- Cada iteração da fase de construção:
  - usando uma função gulosa, avaliar o benefício de cada elemento
  - criar uma lista restrita de candidatos (LRC), formada pelos elementos de melhor avaliação
  - selecionar aleatoriamente um elemento da lista restrita de candidatos
  - adaptar a função gulosa com base na decisão do elemento que foi incluído
- Seleção aleatória é feita entre os melhores elementos da lista de candidatos (não necessariamente o melhor, como na escolha puramente gulosa).
- Diversificação baseada em aleatorização controlada: diferentes soluções construídas em diferentes iterações GRASP

Construção (GRASP)

- Cada iteração da fase de construção:
  - usando uma função gulosa, avaliar o benefício de cada elemento
  - criar uma lista restrita de candidatos (LRC), formada pelos elementos de melhor avaliação
  - selecionar aleatoriamente um elemento da lista restrita de candidatos
  - adaptar a função gulosa com base na decisão do elemento que foi incluído
- Seleção aleatória é feita entre os melhores elementos da lista de candidatos (não necessariamente o melhor, como na escolha puramente gulosa).
- Diversificação baseada em aleatorização controlada: diferentes soluções construídas em diferentes iterações GRASP

Construção (GRASP)

- Cada iteração da fase de construção:
  - usando uma função gulosa, avaliar o benefício de cada elemento
  - criar uma lista restrita de candidatos (LRC), formada pelos elementos de melhor avaliação
  - selecionar aleatoriamente um elemento da lista restrita de candidatos
  - adaptar a função gulosa com base na decisão do elemento que foi incluído
- Seleção aleatória é feita entre os melhores elementos da lista de candidatos (não necessariamente o melhor, como na escolha puramente gulosa).
- Diversificação baseada em aleatorização controlada: diferentes soluções construídas em diferentes iterações GRASP

Construção (GRASP)

$s \leftarrow \emptyset$
repeat
  $C \leftarrow ProduzirLRC()$
  $c \leftarrow SelecionarElementoAleatório(C)$
  $s \leftarrow s \cup c$
until (solução não construída)
return s

Princípio:

A técnica de escolha permite que diferentes soluções sejam geradas em cada iteração GRASP.

Construção (GRASP)

$s \leftarrow \emptyset$
repeat
  $C \leftarrow$ *ProduzirLRC*()
  $c \leftarrow$ *SelecionarElementoAleatório*(C)
  $s \leftarrow s \cup c$
until (solução não construída)
return s

Princípio:

A técnica de escolha permite que diferentes soluções sejam geradas em cada iteração GRASP.

Construção (GRASP)

$s \leftarrow \emptyset$
repeat
  $C \leftarrow$ *ProduzirLRC*()
  $c \leftarrow$ *SelecionarElementoAleatório*(C)
  $s \leftarrow s \cup c$
until (solução não construída)
return s

Princípio:

A técnica de escolha permite que diferentes soluções sejam geradas em cada iteração GRASP.